UNIVERSITA DELLA CALABRIA

Dipartimento di ELETTRONICA, INFORMATICA E SISTEMISTICA

Models and Methods for the Constrained Shortest Path Problem and its variants

PhD thesis in

Operational Research - MAT/09

Coordinator

Prof. Lucio Grandinetti

[illegible]

Supervisor

Prof.ssa Francesca Guerriero

[illegible]

Candidate

Luigi Di Puglia Pugliese

[illegible]

November 2010

UNIVERSITA DELLA CALABRIA

Dipartimento di ELETTRONICA, INFORMATICA E SISTEMISTICA

Modelli e Metodi per il Problema del Cammino Minimo Vincolato e sue varianti

Tesi di dottorato in

Ricerca Operativa - MAT/09

Coordinatore

Prof. Lucio Grandinetti

[illegible]

Supervisore

Prof.ssa Francesca Guerriero

[illegible]

Candidato

Luigi Di Puglia Pugliese

[illegible]

Novembre 2010

Summary in Italian

Il lavoro di tesi sintetizza i risultati piú importanti delle attivitá di ricerca svolte durante i tre anni del corso di dottorato in Ricerca Operativa. Tali attivitá hanno riguardato la definizione di modelli e metodi innovativi per il problema del cammino minimo vincolato (Constrained Shortest Path Problem, $c\mathcal{SPP}$).

Tale problema modella diverse applicazioni reali ed inoltre, i vincoli possono essere di varia natura. Ad esempio, il percorso puó essere vincolato ad includere specifici nodi, o é necessario includere un certo numero di nodi, oppure ancora includere nodi solo se all'interno di una pre-specificata copertura. Altre applicazioni ricadono nella gestione della rete ferroviaria e nei sistemi di gestione degli aeromobili militari. Inoltre, il $c\mathcal{SPP}$ si presenta anche come sotto problema quando approcci a generazione di colonna vengono utilizzati per risolvere alcuni ben noti problemi come la pianificazione del personale di bordo, problemi di routing di aeromobili a lungo raggio e problemi di instradamento dei veicoli con vincoli aggiuntivi (Constrained Vehicle Routing Problem).

Sulla base delle considerazioni precedenti, in questo lavoro sono state studiate le seguenti varianti del problema:

- il cammino minimo con vincoli sulle risorse (resource constrained shortest path problem, $\mathcal{RCSPP}$);
- il cammino minimo con cammini probiti (shortets path problem with forbidden path, $\mathcal{SPPFP}$);

- il cammino di minimo costo per unitá di tempo di tipo elementare con vincoli di finestre temporali (linear fractional elementary shortest path problem with time windows, $\mathcal{LFESPPTW}$);
- il cammino minimo di tipo elementare con vincoli sulle risorse (resource constrained elementary shortest path problem, $\mathcal{RCESPP}$);
- il cammino minimo multi obiettivo con metriche di varia natura e vincoli di tipo soft;
- il cammini minimo in presenza di cicli negativi.

Per ogni problema sono stati definiti approcci innovativi di soluzione, sono state analizzate le proprietá teoriche dei metodi proposti e sono state validate le loro prestazioni in termini di efficienza ed efficacia mediante una articolata fase sperimentale.

Il presente lavoro é organizzato in 9 capitoli. Nel seguito viene riportata una breve descrizione del contenuto di ogni capitolo. Una introduzione al presente lavoro é data nel capitolo 1. Nel capitolo 2 viene affrontato il $\mathcal{RCSPP}$. In particolare, un innovativo metodo di risoluzione é proposto. Inoltre, procedure per la determinazione di lower e upper bounds sul valore ottimo di funzione obiettivo sono definite. In questo lavoro é stata introdotta una nuova formulazione per il $\mathcal{RCSPP}$. Tale formulazione si basa sul concetto di reference point (punto di riferimento). In altre parole, la soluzione ottima é determinata massimizzando una “achievement scalarizing function”. Questo tipo di funzioni misurano la distanza di una soluzione da un punto di riferimento. Variando il punto di riferimento, varia la soluzione ottenuta. Nel lavoro svolto é stata considerata come misura di distanza dal punto di riferimento la metrica di Chebichef. Partendo dalla formulazione proposta, é stato definito un approccio di risoluzione che iterativamente esplora diverse aree dello spazio di ricerca. La dimensione di tali aree e la loro dislocazione spaziale é definita dal punto di riferimento.

Sia il modello matematico che il metodo di risoluzione sono stati studiati dal punto di vista teorico. Risultati sulla correttezza del modello e sulla convergenza all'ottimo del metodo sono stati ricavati ed esposti.

La strategia di risoluzione é stata implementata in ambiente java e testata su reti benchmark presenti nella letteratura. La parte sperimentale é stata condotta per valutare sia il comportamento dell'approccio proposto e sia l'efficienza dello stesso rispetto allo stato dell'arte. Infatti, nella fase computazionale sono stati considerati anche i metodi piú efficienti apparsi recentemente in letteratura. In particolare, sono stati vagliati sia l'approccio risolutivo basato sulla programmazione dinamica ([22]), sia la strategia di risoluzione basata su metodi per la determinazione dei k cammini minimi ([125]).

I risultati ottenuti sono molto promettenti. Il metodo risolutivo proposto risulta essere piú efficiente dei due algoritmi considerati. Inoltre, i modelli ed i metodi sviluppati per determinare valori di upper bounds, migliorano i risultati ottenuti in [22] e forniscono la soluzione ottima per tutte le reti considerate in [125].

I capitoli 3 e 4 sono dedicati al $\mathcal{SPPFP}$. Di tale problema é stata analizzata sia la variante elementare che quella non elementare. É utile osservare che in entrambi i casi il problema é definito su grafi orientati con costi non negativi. La necessitá di dover affrontare anche il caso elementare ($\mathcal{ESPFP}$) deriva dal fatto che la soluzione del $\mathcal{SPPFP}$ puó contenere dei cicli. Tali cicli sono presenti nel cammino ottimo al fine di interrompere la sequenza d'archi proibita ed ottenere una soluzione ammissibile. Nel capitolo 3 si presentano i risultati ottenuto nella fase di studio relativa al $\mathcal{SPPFP}$. Il primo lavoro sul $\mathcal{SPPFP}$ é apparso in letteratura nel 2005 ([138]). Gli autori ne hanno dato una descrizione formale ma non hanno sviluppato alcun modello matematico per la sua rappresentazione. Nello stesso articolo viene proposto un approccio risolutivo basato sulla modifica del grafo originario, effettuata considerando l'intero insieme dei cammini proibiti. Sul nuovo grafo basta applicare un qualsiasi algoritmo del cammino minimo per determinare una soluzione che non contiene alcun cammino proibito. É utile osservare che l'approccio proposto in [138] é un algoritmo polinomiale. Un ulteriore contributo, é stato dato in [30] in cui gli autori propongono delle piccole modifiche all'algoritmo proposto in [138] che permettono di ottenere un certo margine di guadagno in termini computazionali. L'approccio proposto, invece, é di tipo pseudo polinomiale e si basa sulla formulazione dinamica del problema. In particolare, é stato esteso l'algoritmo di Desrochers [50] al caso preso in esame e diverse versioni sono state definite. Lo studio computazionale ha messo in evidenza che in pratica gli approcci proposti risultano essere piú efficienti dell'algoritmo polinomiale ([138]) e della versione migliorata ([30]). Inoltre, le versioni definite risultano piú efficienti dell'algoritmo di base proposto da Desrochers ([50]).

La versione elementare del problema del cammino minimo con cammini proibiti non é stata studiata nella letteratura scientifica. Tale problema é stato approfondito ed i risultati sono riportati nel capitolo 4. In particolare, é stato sviluppato un modello matematico per la rappresentazione del problema e sono stati definiti due approcci di risoluzione: un metodo di Branch e Bound ed un approccio di programmazione dinamica. Nel primo metodo di soluzione il problema viene rilassato eliminando i vincoli sui cammini proibiti in modo da risolvere un classico problema del cammino minimo. Ad ogni sottoproblema viene (o vengono) inserito (o inseriti) il (o i) vincolo (o vincoli) violato (o violati) nella soluzione inammissibile del problema padre, grazie ad appropriate strategie di branching. Tale strategia di risoluzione é stata definita per il caso singola origine singola destinazione. Successivamente l'approccio di risoluzione é stato esteso al caso singola origine a tutte le altre destinazioni generalizzando i risultati teorici ottenuti dallo studio del primo problema. Inoltre, al fine di migliorare le performance dell'algoritmo proposto é stata definita una procedura euristica che consente di costruire una soluzione ammissibile che viene utilizzata per inizializzare il metodo e sono stati introdotti lower bounds con lo scopo di individuare nodi dell'albero non promettenti. Il secondo approccio di risoluzione si basa sulle programmazione dinamica. In particolare, metodi a correzione di contrassegno sono stati definiti con diverse strategie di estrazione sia di nodi che di etichette. Gli approcci proposti consentono di risolvere il caso singola origine e tutte le destinazioni e di conseguenza anche il caso a singola destinazione. La fase di test é stata condotta in maniera tale da considerare due casi limite: 1) tutti i vincoli rendono inammissibile la soluzione del rilassato; 2) una porzione ridotta di vincoli rende inammissibile il problema rilassato. Dai risultati ottenuti si evince che per il primo caso, il secondo metodo risulta essere nettamente superiore in termini di efficienza computazionale. Per il secondo set di istanze, la bontá degli algoritmi dipende dalle dimensioni delle reti. In particolare, per reti di piccola dimensione, l'approccio basato sul metodo del branch and bound risulta piú veloce del metodo basato sulla programmazione dinamica. Per le istanze con numero di nodi maggiore di 400 e numero di archi maggiore di 5000, il secondo metodo di soluzione é piú efficiente.

Nel capitolo 5 viene presentato il lavoro svolto relativamente al $\mathcal{LFESPPTW}$. Tale problema é definito su grafi con informazione sia sul costo che sul tempo di attraversamento degli archi. L'obiettivo é la minimizzazione del costo per unitá di tempo da un nodo origine ad un nodo destinazione. É evidente che la definizione della funzione obiettivo implica la possibilitá di ottenere una soluzione ciclica.

Inoltre, ad ogni nodo della rete é associata una finestra temporale che indica il lasso di tempo in cui tale nodo puó essere visitato. I vincoli di finestra temporale sono di tipo hard, in altre parole non é ammessa alcuna attesa ed inoltre non é possibile visitare il nodo se il tempo di arrivo allo stesso supera il tempo massimo ammissibile di servizio.

Il problema é stato formulato con un modello di programmazione lineare binaria. L'approccio di risoluzione proposto si basa sullo schema algoritmico a correzione di contrassegno, in cui un insieme di etichette multi dimensionali é associato a ciascun nodo della rete. Sono stati sviluppati metodi di tipo forward e bi-direzionale, che considerano diverse strategie di estrazione sia di etichette che di nodi. Accanto al ben noto criterio di dominanza paretiana, che in questo contesto é stato opportunamente modificato, si é definito un nuovo criterio di dominanza. Tale criterio é applicabile solo dopo l'esecuzione di una innovativa strategia di preprocessamento.

É utile osservare che la letteratura scientifica non aveva ancora considerato tale problema. La fase di test é stata condotta con lo scopo di valutare le diverse versioni e l'efficienza dei criteri specifici di dominanza introdotti per il $\mathcal{LFESPPTW}$. I risultati sono stati collezionati sia su reti random, cicliche ed acicliche e sia sulle reti benchmark di Solomon.

Per quanto riguarda il $\mathcal{RCESPP}$, nela capitolo 6 viene riportata una dettagliata analisi dello stato dell'arte, enfatizzando le strategie di risoluzione. Due sono i contributi di tale lavoro. In primo luogo viene proposto un algoritmo prototipo dal quale é possibile derivare i metodi al momento piú efficienti per la risoluzione del $\mathcal{RCESPP}$. Inoltre, per la prima volta, vengono analizzati e comparati i metodi piú efficienti proposti nella letteratura scientifica per la risoluzione del $\mathcal{RCESPP}$. Le strategie di risoluzione sono apparse nello stesso periodo e si basano su concetti simili. Il lavoro é stato anche quello di analizzare i punti in comune degli algoritmi e di evidenziarne gli aspetti connotativi di ciascuno. É utile osservare, che i metodi di risoluzione sono stati testati dai rispettivi autori su reti differenti. Nel lavoro presentato in questa tesi, la fase di test é stata unificata e gli algoritmi sono stati implementati utilizzando le stesse strutture dati. In questo modo é stato possibile fare un confronto su un piú ampio set di problemi test. I risultati ottenuti mostrano una non dominanza fra gli algoritmi. Inoltre, interessanti considerazioni sono state ricavate in merito allo sforzo computazionale degli algoritmi rispetto alla variazione della regione ammissibile.

Nel capitolo 7 si presentano i risultati ottenuti relativamente alla linea di ricerca che ha riguardato l'analisi multi obiettivo e la definizione di algoritmi di instradamento vincolato su reti di telecomunicazione. Il problema che si é risolto é quello di determinare la soluzione che piú si avvicina ai desideri del “decision maker” nel caso in cui importanti parametri che caratterizzano le reti di telecomunicazione quali costo, ritardo, jitter, bandwidth, packet loss ed error rate sono considerati. In particolare, ciascuno di questi parametri, non necessariamente di tipo additivo, é vincolato a mantenersi in un determinato intervallo/range, che il decisore esprime come preferenze. É utile osservare che tale problema non é mai stato affrontato nella letteratura scientifica, nonostante risulta essere di elevata importanza pratica. Infatti la definizione dello stesso é stata possibile grazie alla visita effettuata al National Institute of Telecommunication di Varsavia. Il problema é stato formulato considerando il concetto di reference point. In tale modo si é potuto modellare i range per ciascun criterio come vincoli di tipo soft. In altre parole, se una soluzione tale per cui ciascun criterio rientri nel range specificato dal “decision maker” non esiste, il metodo restituisce una soluzione tale per cui ciascun criterio sia il piú vicino possibile all'intervallo dato. Il modello matematico é risolto attraverso un approccio risolutivo che fornisce la soluzione ottima al problema, ovvero la soluzione che piú si avvicina alle preferenze del decision maker. In particolare, l'ottimalitá della soluzione ammissibile, determinata attraverso un algoritmo a correzione di contrassegno viene valutata attraverso uno schema di branching che assicura una esplorazione esaustiva dello spazio di ricerca. Sono state analizzate le proprietá teoriche del metodo proposto. Inoltre, una estesa fase computazionale é stata condotta con lo scopo di valutare le performance dell'algoritmo proposto rispetto a diversi scenari. In altre parole, sono state considerate sia reti random, sia reti complete e sia reti a griglia. Inoltre, diverse istanze sono state ricavate simulando diverse scelte del decision maker. I risultati mostrano come l'approccio proposto risolve tutte le istanze generate in tempi ragionevoli e le prestazioni dello stesso dipendono molto dall'ampiezza del range definito per ciascun criterio.

Nel capitolo 8 il problema del cammino minimo elementare é considerato. Cicli di costo negativo sono presenti nel grafo. L'obiettivo é di determinare i cammini di costo minimo aciclici da un nodo sorgente a tutti gli altri nodi del grafo. Per tale problema é stato sviluppato un modello matematico che é costituito dai vincoli di conservazione del flusso classici della rappresentazione matemetica del problema non vincolato ed i vincoli di eliminazione dei sotto cicli. Tre differenti strategie di risoluzione sono state definite. L'idea alla base dei metodi proposti é la determinazione del minor numero di cammini che é necessario determinare per ciascun nodo al fine di ottenere i cammini minimi aciclici per tutti i nodi del grafo. Il primo approccio considerato si basa sulla formulazione dinamica del problema. In particolare, ulteriori risorse fittizie vengono aggiunte ai nodi dai queli si generano i cicli di costo negativo. Le risorse fittizie mentengono traccia della visita del nodo a cui sono associate lungo un qualsiasi cammino. Il secondo metodo si basa sulla determinazione dei primi k cammini minimi. In tale contesto il valore di k differisce da nodo a nodo ed inoltre tale parametro é una variabile. L'ultimo approccio é un metodo di Branch e Bound. Tale strategia di risoluzione si basa sulla determinazione di upper bounds. Ogni qualvolta un ciclo di costo negativo viene individuato, un certo insieme di sotto grafi viene generato attraverso appropriate strategie di branching. In tali sotto grafi il ciclo individuato viene eliminato.

I metodi proposti sono stati analizzati dal punto di vista teorico e sono stati comparati considerando la complessitá computazionale nel caso peggiore.

Nel capitolo 9 vengono riportate le conclusioni del lavoro.

Contents

Part I

Introduction

Chapter 1

Introduction

In this thesis we present the main results achieved in the three years of the PhD program. The research activities have been focused in developing and designing innovative models and methods for the Constrained Shortest Path Problem ($c\mathcal{SPP}$, for short).

1.1 Motivation

The $c\mathcal{SPP}$ is an extension of the Shortest Path Problem ($\mathcal{SPP}$, for short) in which the optimal path is constrained to respect some feasibility criterion. While the former is easy to solve, the introduction of further constraints makes the $\mathcal{SPP}$ a NP-hard problem. Despite to its theoretical complexity and for its practical importance, many studies have been carried out in order to efficiently solve the $c\mathcal{SPP}$. Indeed, the $\mathcal{SPP}$ and its constrained counterpart are used to model several real-life applications. Several constrained versions of the basic $\mathcal{SPP}$ exist. For example, the path may be constrained to include specific nodes, or be constrained to include a specific number of nodes, or include nodes within a pre-specified covering distance of every node in the network.

Another type of constraint is that establishes an upper limit on the sum of some other arc cost. This problem is known as Weight-

Constrained Shortest Path Problem ($\mathcal{WCSPP}$, for short). It is important to point out that when more than one of such a constraints are included, the scientific literature refers to the problem as Resource Constrained Shortest Path Problem ($\mathcal{RCSPP}$, for short).

The $\mathcal{WCSPP}$ and the $\mathcal{RCSPP}$ apply to a number of real-world situations, i.e. railroad management, military aircraft management systems, routing in road networks. The $\mathcal{WCSPP}$ and the $\mathcal{RCSPP}$ also arise in practice when column-generation approaches are used to solve some well-known problems like dated crew scheduling, day-of-operations rescheduling activities, or long-haul aircraft routing problems.

When the column-generation approach is used to solve the Vehicle Routing Problem with additional Constraints, the pricing problem is formulated as a $\mathcal{RCSPP}$. In this case, the most common constraints are related to the capacity of the vehicles and to specific requests of the costomer, that is time windows can be associated with each node and refer to the time renge in which the related constumers can be served. However, in the pricing problem the arc costs may be negative and the graph can contain cycles. In this case, the aim is to determine the optimal path with no repeated nodes. This means that an elementary and feasible path with minimum cost must be found. The scientific literature refers to this problem as Resource Constrained Elementary Shortest Path Problem ($\mathcal{RCESPP}$, for short).

The $\mathcal{SPP}$ with forbidden path ($\mathcal{SPPFP}$, for short) have been appeared quite recently in the scintific literature. Also this instance of the $c\mathcal{SPP}$ arises when a column-generation approach is used to solve more difficult problems. In this case, the optimal path does not include specific sequences of arcs, that is forbidden paths.

1.2 Goals

The scientific literature provides several solution approaches to solve the different instances of the $c\mathcal{SPP}$. The most common strategies used to optimally solve the problems are: 1) the dinaming programming approach; 2) methods based on path ranking and 3) those based on branch-and-bound scheme. It is worth observing that in most cases the aforementioned strategies are used to close the duality gap obtained by solving a relaxation of the problem. The constraints are relaxed in Lagreangian objective function. This technique allows to treat the $c\mathcal{SPP}$ as a $\mathcal{SPP}$, thus efficient methods are available to solve the Lagreangian relaxation. It is important to point out that the Lagreangian relaxation is useful only if no negative cost cycles are present. When the arc costs are not constrained in sign, others types of relaxation are used.

The main aim of this thesis is the definition, the design, the implementation and to computationally test innovative models and methods for different instances of the $c\mathcal{SPP}$.

1.3 Contribution

In this thesis we propose innovative models and methods to addres the following instances: 1) the Resource Constrained Shortest Path Problem; 2) the Shortest Path Problem with Forbidden Paths; 3) the Elementary version of the Shortest Path Problem with Forbidden Paths ($\mathcal{ESPFP}$, for short); 4) the Linear Franctional Elementary Shortest Path Problem with Time Windows ($\mathcal{LFESPPTW}$, for short); 5) the Multi-criteria Path Problem with multiple metrics and Soft Constraints ($\mathcal{MPPSC}$, for short); 6) the Elementary Shortest Path Problem ($\mathcal{ESPP}$, for short).

The methods defined for the aforementioned instances are based both on the dynamic programming and branch-and-bound strategies.

The behaviour of the proposed solution approaches has been evaluated on a wide range of test problems both taken from the literature and generated by ourself. The tests have been carried out in an intensive, appropriate and extensive computational phase and the innovative models and methods have been compared with the best solution approaches present in the scientific literature to addres the considered instances of the $c\mathcal{SPP}$.

It is worth observing that in this thesis we have also investigated instances that are not yet studied in the scientific literature. We refer to the Elementary version of the $\mathcal{SPPFP}$, the $\mathcal{LFESPPTW}$, the $\mathcal{MPPSC}$ and the $\mathcal{ESPP}$.

In addition, a detailed study on the $\mathcal{RCESPP}$ is presented and it is focused on the resolution approaches proposed in the scientific literature. In particular, for the first time a computational study has been carried out in order to empirically evaluate the behaviour of the strategies that turn out to be the most efficient to solve the $\mathcal{RCESPP}$.

1.4 Organization of the thesis

The thesis is organized as follows.

In chapter 2 the results achieved on the studies devoted to the $\mathcal{RCSPP}$ are presented. An innovative strategy based on the Reference Point Method is defined. Upper and lower bounds for the $\mathcal{RCSPP}$ are proposed. The results are very encouraging. The proposed method turn out to be competitive with the state of the art approaches and the obtained lower and upper bounds are better than those proposed in the scientific literature.

Chapter 3 and 4 are devoted to the $\mathcal{SPPFP}$ and its elementary counterpart, respectively. For the first problem, dynamic programming approaches have been defined and they are based on the Desrochers' algorithm. The computational analysis suggests that the proposed versions outperform the naive counterpart and the state of the art strategies, based on graph modification, result less efficient than the proposed dynamic programming methods. In chapter 4 the Elementary $\mathcal{SPPFP}$, that is $\mathcal{ESPFP}$ is addressed. It is important to point out that this study represent the first attempt to define solution approaches for the $\mathcal{ESPFP}$. Two strategies have been proposed: 1) dynamic programming approach and 2) branch-and-bound scheme. Different versions of dynamic programming approach have been designed. They are based on both node and state selection methods. In addition, several selection strategies have been considered. Regarding the branch-and-bound approach, also in this case several selection strategies of the next problem, that have to be examined, have been defined. In addition, an ad-hoc heuristic procedure is devised in order to construct a feasible solution for initializing the proposed branch-and-bounbd methods. The performances of the naive methods have been improved by considering fathoming rule that makes use of appropriate lower bounds.

The $\mathcal{LFESPPTW}$ is adderessed in chapter 5. Multi-dimensional labelling algorithms are proposed to solve this variant of the classical $\mathcal{SPP}$. Extensive computational tests are carried out on a meaningful number of test problems, with the goal of assessing the behaviour of the proposed approaches. The computational study shows that the introduction of dominance rules and the adoption of a bi-directional search strategy allow the definition of solution approaches that turn out to be very effective in solving the $\mathcal{LFESPPTW}$.

In chapter 6, a computational study of the most efficient solution approaches for the $\mathcal{RCESPP}$ is presented. In addition, a prototype framework is proposed from which the studied solution strategies can be derived.

The studies related to the instace with multiple metrics and soft constraints are presented in chapter 7. We provide a description of the problem and we propose a formulation based on the Chebichef distance. A branch-and-bound approach is devised and a computational study has been carried out on different types of networks and scenarious.

In chapter 8 the $\mathcal{ESPP}$ is addressed. Three strategies have been developed to solve the problem under investigation. The first is based on the dynamic programming formulation, the second method determines the first k shortest paths where k is considered as a variable. In the last approach, upper bounds are computed and the optimality gap is closed by using a branch and bound scheme. The proposed algorithms are theoretically evaluated.

Conclusions are given in chapter 9.

Part II

Constrained Shortest Path

Chapter 2

Reference point based solution approach for the resources constrained shortest path problem [1] [2]

Luigi Di Puglia Pugliese

Department of Electronics, Informatics and Systems, University of Calabria, 87036, Rende (CS), Italy

Francesca Guerriero

Department of Electronics, Informatics and Systems, University of Calabria, 87036, Rende (CS), Italy

Abstract

The Resources Constrained Shortest Path Problem ($\mathcal{RCSPP}$) is a variant of the classical shortest path problem of a great practical importance. The aim is to find the shortest path between a given pair

[1]Submitted for pubblication in the journal *Transportation Science*.

[2]Work presented at the EURO XXIV, the 24th European Conference on Operational Research, Lisbon, July 11-14, 2010

of nodes, under additional constraints, representing upper bounds on the consumption of resources along the path. In the scientific literature, different approaches have been defined to solve the $\mathcal{RCSPP}$. In this work we propose an innovative interactive method to address the $\mathcal{RCSPP}$, based on a novel search strategy of the criteria space. The performance of the proposed approach is evaluated on the basis of an extensive computational study, by considering benckmark instances. A comparison with the state-of-art approaches developed for the $\mathcal{RCSPP}$ is also carried out. The computational results have shown that the developed solution strategy is competitive with the most efficient strategies known so far.

Keywords: constrained shortest paths; interactive method; label-Correcting method; lower and upper bounds.

2.1 Introduction

The Resources Constrained Shortest Path Problem, in the sequel referred to as $\mathcal{RCSPP}$, aims at finding the shortest path from a pre-determined source node to a pre-determined destination node, through a directed graph with cost and resource consumptions associated with each arc, for which the total consumption of each resource along the path is less than or equal to a given upper bound.

It is worth observing that, in the scientific literature, the problem, where only one resource is considered, is often referred to as the weigth constrained shortest path problem ($\mathcal{WCSPP}$, for short).

The addition of such constraints to the classical shortest path problem generally results in a problem that belongs to the $\mathcal{NP}$-hard class ([71], [81], [91]). Dumitrescu and Boland indicate that the $\mathcal{WCSPP}$ is $\mathcal{NP}$-hard even if the graph is acyclic and all resources and costs are positive ([60]). If the direct graph does not contain negative cost cycles, the problem is $\mathcal{NP}$-hard in the weak sense and it can be solved in pseudo-polynomial time. In addition, it is polynomial solvable if either the costs or the resource requirements are bounded ([84]). The $\mathcal{RCSPP}$ also belongs to the $\mathcal{NP}$-hard class and it can be solved in pseudo-polynomial time for a fixed number of resources and if negative cost cycles are not present.

Several real-world situations can be mathematically formulated as a $\mathcal{RCSPP}$. Zabarankin et al. [149] described an application to military aircraft management systems, whereas Halpern and Priess [80] presented an application of the $\mathcal{RCSPP}$ to railroad management. Other application areas include the waste water management ([61]), the quality of service routing in communications networks ([145]) and linear curve approximation ([113]).

The $\mathcal{RCSPP}$ also arises in practice when column-generation approaches are used to solve some well-known problems like dated crew scheduling, day-of-operations rescheduling activities, or long-haul aircraft routing problems [13]. In addition, it is also a common substructure in difficult optimization problems (i.e., fleet management [7], dial-a-flight problems [63]).

A large number of exact and approximate methods have been developed to address the $\mathcal{RCSPP}$ and the $\mathcal{WCSPP}$. The solution procedures can be grouped in the following main classes: 1) path ranking methods; 2) Lagrangean relaxation methods; 3) node labeling methods. In addition, some solution strategies have been defined by combining the approaches belonging to the aforementioned categories.

A path ranking method to address the $\mathcal{WCSPP}$ has been proposed in [81]. The main idea is to determine k shortest paths from the origin node to the destination node, until a feasible path is identified. The main drawback of this approach is that a large value for k is required, before a feasible solution of the $\mathcal{WCSPP}$ is found. Very recently, Santos et al. ([125]) propose an improved solution algorithm based on the solution of the k-shortest paths problem that outperforms the approaches of [81]. The main difference is related to the way in which the k shortest paths are sorted.

Many scientific contributions consider the Lagrangean relaxation as a viable alternative to solve the $\mathcal{RCSPP}$ and the $\mathcal{WCSPP}$. Indeed, if the resource constraints are relaxed, the resulting subproblem becomes an instance of the classical shortest path problem, that can be solved repeatedly with a very limited computational effort.

The approaches based on Lagrangean relaxation rely on the solution of a Lagrangean dual problem and consider different strategies to close the duality gap. Handler and Zang in [81] presented a specialized version of the Kelley's cutting plane to solve the Lagrangean dual problem associated to the $\mathcal{WCSPP}$ and closed the duality gap by applying the k-shortest paths algorithm of [146], with edge lengths set to the reduced costs arising from the optimal Lagrangean dual solution. An approach similar to the one considered in [81] has been proposed by Carlyle and Wood ([26]). In this case, the duality gap is closed by using a depth first branch-and-bound and a bisection search is adopted to determine the optimal Lagrange multiplier.

Beasley and Christofides [16] solved the dual of the $\mathcal{RCSPP}$ with subgradient optimization and applied a branch-and-bound procedure to close the duality gap (Lagrangean dual values are used as lower bounds at each node of the tree). Mehlhorn and Ziegelmann [105] proposed a hull approach to solve a linear relaxation of the $\mathcal{RCSPP}$ and considered three differet methods to close the gap between the lower and the upper bounds: the algorithm of Hassin ([84]), the k-shortest paths algorithm of [93] and a dynamic programming algorithm (of a label-correcting type) that uses path ranking and prunes paths that cannot be extended to an optimal solution.

It is worth observing that relaxation has also been used in heuristic approaches. We cite, for example, the paper of Ribeiro and Minoux ([118]), where a Lagrangean relaxation approach to address a generalization of the $\mathcal{WCSPP}$ with upper and lower resource limits is proposed and the paper of Avella et al. ([7]), where the resource constraints of the $\mathcal{RCSPP}$ are relaxed into an exponential penalty function.

Starting from the pioneering works of [94] and [97], where dynamic programming formulations for the $\mathcal{WCSPP}$ have been proposed, intense research activity has been carried out in the development of node labeling solution approaches based on these formulations. We cite, for example, the methods of Desrosiers et al. [53], Desrochers and Soumis [52] and Jaumard et al. [92]. The label setting algorithm of Desrochers and Soumis ([52]) seems to be the most effective approach for the $\mathcal{WCSPP}$. Dumitrescu and Boland in [60] proposed a modified version of this algorithm which uses preprocessing and Lagrange multiplier information. Very recently, Muhandiramge and Boland ([111]) have defined a new preprocessing procedure for the $\mathcal{WCSPP}$ in which Lagrangian relaxation technique and network reduction steps are tightly integrated.

Addressing the $\mathcal{RCSPP}$ with a dynamic programming approach is also closely related to the solution of multi-objective shortest path problems, which have received the attention of many researchers. The aim in these problems is to generate non dominated paths (i.e., Pareto optimal paths). Several studies have focused on the development of efficient solution approaches to address the multi-criteria shortest path problem. They can be classified in four main categories. The first group includes exact procedures such as multiple labelling methods (e.g., [79], [103], [128]); ranking methods (e.g., [10], [36]); parametric methods (e.g., [110]). The second group contains the approximate procedures (e.g., [85], [135], [142]). The third class utilizes utility (or cost) functions (e.g., [27], [107], [108]). The last group includes the interactive methods (e.g., [40], [74], [112]).

Reference point methodology provides the theoretical foundations for many interactive search procedures in multiple objective optimization (for example, [143], [144]). A reference point consists of desirable values for each objective function. In multiple objective mathematical programming, solution methods based on reference points can generate non dominated solutions using a variety of scalarizing functions ([25]) that define the distance from the reference point. The idea is to take the desires of the decision maker into account when projecting the reference point onto the set of non dominated solutions. In several works the reference point methodology has been applied to find a particular non dominated solution in continuous search space (e.g., [114], [124], [98]). In [74] the reference point method is used to define a solution approach, based on label correcting procedure, that allows to find a properly Pareto optimal solution when solving the multi-objective shortest path problem.

In this paper, we deal with the solution of the $\mathcal{RCSPP}$ and we propose an interactive procedure, that is based on the reference point methodology of Wierzbicki ([143], [144]). The main idea is to exploit the solution space, by defining a sequence of reference points. To the best of our knowledge, this work represents the first attempt to define a reference point method to address the $\mathcal{RCSPP}$. The contribution of this paper is three fold. First of all, an innovative solution method is defined to solve the $\mathcal{RCSPP}$ and new models, based on reference point methodology, are proposed to compute upper and lower bounds for the problem at hand. Second, theoretical aspects related to the models and methods proposed in this paper are investigated. Finally, a computational study is carried out with the aim of comparing the proposed method with the state-of-art algorithms presented in the scientific literature to address the $\mathcal{RCSPP}$.

The rest of the paper is organized as follows. In Section 2, we give a description of the problem at hand, the proposed solution method is presented in Section 3. In Section 4, we compare computationally the devised approach with the methods of Dumitrescu and Boland [60] and Santos et al. ([125]), that turn out to be the most efficient approaches to address the problem under study. Section 5 summarizes our conclusions and discusses future work.

2.2 Problem formulation

The $\mathcal{RCSPP}$ is defined on a directed graph $\mathcal{D} = (\mathcal{N}, \mathcal{A})$, where $\mathcal{N} = \{1, \dots, n\}$ denotes the set of nodes, and $\mathcal{A} \subseteq \mathcal{N} \times \mathcal{N}$ the set of m arcs. It is assumed that the directed graph $\mathcal{D}$ is *simple*, that is at most one arc connects any two nodes. With each arc (i, j) are associated an integer scalar $c_{ij} \geq 0$, that represents the cost to traverse the arc (i, j) and a vector $w_{ij} = (w^1_{ij}, \dots, w^p_{ij})$ containing information about the resources consumption along the arc (i, j). It is assumed that $w^k_{ij} \geq 0$ and integers $\forall (i, j) \in \mathcal{A}$, $k = 1, \dots, p$. We use the vector $q^{(i,j)} \in Z^{p+1}$ to store the parameters associated with the arc (i, j).

Given two distinct nodes i and j, a path Π_{ij} from node i to node j is a sequence of nodes Π_{ij}=$\{i = i_1, \dots, i_l = j\}$, $l \geq 2$ such that $(i_h, i_{h+1}) \in A$, for h=$1, \dots, l-1$.

We refer to the cost of a path Π_{uv} as $c(\Pi_{uv})$, representing the sum of the cost c_{ij} associated with the arcs that belong to Π_{uv}, i.e., $c(\Pi_{uv})$=$\sum_{(i,j)\in\Pi_{uv}} c_{ij}$. For each resource k, $k = 1, \dots, p$, we indicate with $w^k(\Pi_{uv}) = \sum_{(i,j)\in\Pi_{uv}} w^k_{ij}$ the quantity of the resource consumed along the path Π_{uv}. We define a vector $q(\Pi_{uv}) = c(\Pi_{uv}) \cup w(\Pi_{uv}) \in Z^{p+1}$, where $w(\Pi_{uv}) = [w^1(\Pi_{uv}), \dots, w^p(\Pi_{uv})]^T$.

The $\mathcal{RCSPP}$ consists in finding the least cost path in $\mathcal{D}$ from a specified source node s to a destination node d, such that the consumption of each resource k=$1, \dots, p$ is less than or equal to a specific upper bound $\mathcal{W}^k$.

Let $\bar{\Pi}$ be the set of all paths from node s to node d, the $\mathcal{RCSPP}$ can be mathematically represented as follow:

$$\min_{\Pi \in \bar{\Pi}} c(\Pi) \tag{2.1}$$

$$s.t.$$

$$w^k(\Pi) \leq W^k, \quad \forall k = 1, \ldots, p \tag{2.2}$$

In this paper, we propose a solution approach, based on interactive strategy, to address the $\mathcal{RCSPP}$ stated above. In a multi-criteria problem, the decision maker chooses the most satisfactory solution among those that belong to the Pareto optimal set. In many cases, the decision maker has some idea about the characteristics of the solution that he/she wants to achive. For example he/she could have some preferences about the value of each criterion. An interactive method takes into account these preferences, in order to obtain the most satisfactory solution. This selection is implicitly determined by converting a multi-objective problem into a parametric single-objective problem. The selection of a particular properly Pareto optimal path is determined by the definition of the reference point (i.e., aspiration point). Most of those methods use the maximization of an achievement scalarising function (ASF, for short) in the form:

$$S(q, \bar{q}, y) = \min_{0 \leq t \leq p} \{y_t(\bar{q}_t - q_t)\} + \epsilon \sum_{t=0}^{p} y_t(\bar{q}_t - q_t) \tag{2.3}$$

where $\bar{q} \in R^{p+1}$ is an aspiration point, $y_t > 0$, $t = 0, \ldots, p$, are scaling coefficient and ϵ is a given small positive number. Maximization is over all paths from the origin node s to the destination node d.

For a non attainable $\bar{q}$, the vector q of the resulting properly Pareto optimal path is the nearest, in the sense of a Chebyshev weighted norm, to the specific aspiration point $\bar{q}$. If $\bar{q}$ is attainable, then the vector q of the properly Pareto optimal path is uniformly better.

The $\mathcal{RCSPP}$ can be formulated using the ASF (2.3) as follows:

$$\max_{\Pi \in \bar{\Pi}_0} \min_{0 \le t \le p} \left\{ y_t[\bar{q}_t^{opt} - q_t(\Pi)] \right\} + \epsilon \sum_{t=0}^{p} y_t[\bar{q}_t^{opt} - q_t(\Pi)] \tag{2.4}$$

where the set $\bar{\Pi}_0$ contains all the paths Π_{sd} such that $w^k(\Pi_{sd}) \le W^k$, $\bar{q}_0^{opt} = c(\Pi_{sd}^c)$ with $\Pi_{sd}^c{=}arg\min_{\Pi \in \bar{\Pi}} c(\Pi)$, i.e., Π_{sd}^c is the least cost path and $\bar{q}_k^{opt} = W^k + 1, k = 1, \dots, p$.

In the following, we assume that $y_t = 1,\ t = 0, \dots, p$ and $\epsilon = 0$. The proposed formulation for the $\mathcal{RCSPP}$ is equivalent to (2.1) - (2.2). This result is formally proven in the Theorem below.

Theorem 2.2.1. *Problem (2.4) is equivalent to problem (2.1) - (2.2), that is the optimal solution of (2.4) is also optimal for (2.1) - (2.2) and viceversa.*

1. *Let Π_{sd}^* be the optimal solution of problem (2.1) - (2.2), that is $w^k(\Pi_{sd}^*) \le W^k, \forall k = 1, \dots, p$ and $c(\Pi_{sd}^*) \le c(\bar{\Pi}_{sd}), \forall \bar{\Pi}_{sd} \in \bar{\Pi}_0$. Thus, Π_{sd}^* is the optimal solution of problem (2.4), that is*

$$S(q(\Pi_{sd}^*), \bar{q}^{opt}, y) \ge S(q(\bar{\Pi}_{sd}), \bar{q}^{opt}, y),\ \forall \bar{\Pi}_{sd} \in \bar{\Pi}_0. \tag{2.5}$$

2. *Let Π_{sd}^* be the optimal solution of problem (2.4), that is condition (2.5) is satisfied. Thus, Π_{sd}^* is the optimal solution of problem (2.1) - (2.2), that is $w^k(\Pi_{sd}^*) \le W^k, \forall k = 1, \dots, p$ and $c(\Pi_{sd}^*) \le c(\bar{\Pi}_{sd}), \forall \bar{\Pi}_{sd} \in \bar{\Pi}_0$.*

Proof. Since equation (2.2) is satisfied by the path Π_{sd}^* and by each path $\bar{\Pi}_{sd} \in \bar{\Pi}_0$ and in addition, $c(\Pi_{sd}^c) \le c(\bar{\Pi}_{sd})$, $\bar{q}_k^{opt} > W^k$, $\forall k = 1, \dots, p$, we have that

$$\bar{q}_k^{opt} - q_k(\bar{\Pi}_{sd}) > 0, \quad k = 1, \dots, p; \tag{2.6}$$

and

$$\bar{q}_0^{opt} - q_0(\bar{\Pi}_{sd}) \leq 0. \tag{2.7}$$

We will prove part **1.** of the Theorem by contradiction. Thus, let us suppose (for the purposes of contradiction) that there exists a solution $\tilde{\Pi}_{sd} \neq \Pi^*_{sd}$, such that $\tilde{\Pi}_{sd} \in \bar{\Pi}_0$ and it is optimal for (2.4), that is

$$S(q(\tilde{\Pi}_{sd}), \bar{q}^{opt}, y) \geq S(q(\Pi^*_{sd}), \bar{q}^{opt}, y) \equiv \min_{0 \leq t \leq p} \left\{ \bar{q}_t^{opt} - q_t(\tilde{\Pi}_{sd}) \right\} \geq$$

$$\min_{0 \leq t \leq p} \left\{ \bar{q}_t^{opt} - q_t(\Pi^*_{sd}) \right\} \tag{2.8}$$

From equations (2.6) and (2.7), we have that

$$\min_{0 \leq t \leq p} \left\{ \bar{q}_t^{opt} - q_t(\tilde{\Pi}_{sd}) \right\} = c(\Pi^c_{sd}) - c(\tilde{\Pi}_{sd});$$

and

$$\min_{0 \leq t \leq p} \left\{ \bar{q}_t^{opt} - q_t(\Pi^*_{sd}) \right\} = c(\Pi^c_{sd}) - c(\Pi^*_{sd}).$$

Condition (2.8) can be rewritten as follows:

$$c(\Pi^c_{sd}) - c(\tilde{\Pi}_{sd}) \geq c(\Pi^c_{sd}) - c(\Pi^*_{sd}). \tag{2.9}$$

From condition (2.9), we have that $c(\Pi^*_{sd}) \geq c(\tilde{\Pi}_{sd})$, this contradicts the optimality hypothesis of Π^*_{sd} for problem (2.4)

We will prove part **2.** of the Theorem by contradiction. Thus, let us suppose (for the purposes of contradiction) that there exists a solution $\tilde{\Pi}_{sd} \neq \Pi^*_{sd}$, that is optimal for problem (2.1) - (2.2), that is

$$w^k(\tilde{\Pi}_{sd}) \leq W^k, \forall k = 1, \ldots p;$$

$$c(\tilde{\Pi}_{sd}) \leq c(\bar{\Pi}_{sd}), \forall \bar{\Pi}_{sd} \in \bar{\Pi}_0.$$

We have that:

$$S(q(\tilde{\Pi}_{sd}), \bar{q}^{opt}, y) \equiv c(\Pi^c_{sd}) - c(\tilde{\Pi}_{sd});$$

and

$$S(q(\Pi^*_{sd}), \bar{q}^{opt}, y) \equiv c(\Pi^c_{sd}) - c(\Pi^*_{sd}).$$

Since Π^*_{sd} is the optimal solution for problem (2.4), the following condition holds:

$$c(\Pi^c_{sd}) - c(\Pi^*_{sd}) \geq c(\Pi^c_{sd}) - c(\tilde{\Pi}_{sd}). \tag{2.10}$$

Since $c(\tilde{\Pi}_{sd}) \geq c(\Pi^*_{sd})$, this contradicts the optimality hypothesis of $\tilde{\Pi}_{sd}$ for problem (2.1) - (2.2). $\square$

In the next section we define the proposed solution approach to optimality solve the $\mathcal{RCSPP}$.

2.3 Proposed solution approach

In [74], the authors proposed a general label correcting scheme to determine a path on $\mathcal{D}$ for which the function (2.3) is maximized. In our case, the maximization is over a subset of paths from node s to node d. Consequently, the label correcting method proposed in [74] cannot be applied to the problem under consideration, since it could identify a solution that could not be feasible.

The solution approach, proposed in this paper, uses the method presented in [74] to exploit the Pareto front with the aim of identifying a feasible solution with the minimum cost. The search procedure is done by applying a modification of the reference point. In particular, a vector δ is used to perturb the current reference point $\bar{q}$. A new point in the Pareto front is determined by solving the problem reported below:

$$P(\delta) \quad \max_{\Pi \in \bar{\Pi}} \min_{0 \leq t \leq p} \{(\bar{q}_t + \delta_t) - q_t(\Pi)\}; \tag{2.11}$$

where the vector δ is defined in such a way that $\bar{q}+\delta = [UB, W^1, \ldots, W^p]^T$ and UB is the cost of the best feasible solution determined so far. Let $SUB(\delta)$ be a subregion of the criteria space bounded by $\bar{q} + \delta$. If $SUB(\delta) \neq \emptyset$, then the optimal solution of $P(\delta)$ belongs to $SUB(\delta)$. This result is formalized in the following theorem.

Theorem 2.3.1. *If a solution exists in subregion $SUB(\delta)$, then it is the optimal solution of problem $P(\delta)$.*

Proof. Let $\tilde{\Pi}_{sd}$ be a path such that $0 \leq c(\tilde{\Pi}_{sd}) \leq UB$ and $w(\tilde{\Pi}_{sd}) \leq W$, that is $q(\tilde{\Pi}_{sd}) \in SUB(\delta)$.

We prove the theorem by contradiction. In particular, let us suppose that a path $\hat{\Pi}_{sd}$, such that $q(\hat{\Pi}_{sd}) \notin SUB(\delta)$, exists and that $\hat{\Pi}_{sd}$ is the optimal solution of $P(\delta)$. The following inequality holds:

$$S(q(\hat{\Pi}_{sd}), \bar{q} + \delta, y) \geq S(q(\tilde{\Pi}_{sd}), \bar{q} + \delta, y), \tag{2.12}$$

that is

$$\min_{0 \leq t \leq p} \left\{(\bar{q}_t + \delta_t) - q_t(\hat{\Pi}_{sd})\right\} \geq \min_{0 \leq t \leq p} \left\{(\bar{q}_t + \delta_t) - q_t(\tilde{\Pi}_{sd})\right\}. \tag{2.13}$$

Since $q(\hat{\Pi}_{sd}) \notin SUB(\delta)$, the following three situations can occur:

Case 1: $c(\hat{\Pi}_{sd}) > UB$ and $w^k(\hat{\Pi}_{sd}) \leq W^k$;

Case 2: $c(\hat{\Pi}_{sd}) \leq UB$ and $w^k(\hat{\Pi}_{sd}) > W^k$;

Case 3: $c(\hat{\Pi}_{sd}) > UB$ and $w^k(\hat{\Pi}_{sd}) > W^k$.

In the first situation, we have $UB - q_0(\hat{\Pi}_{sd}) < 0$ and $W^k - q_k(\hat{\Pi}_{sd}) \leq\geq 0$. For the second case, $W^k - q_k(\hat{\Pi}_{sd}) < 0$ for some k and $UB - q_0(\hat{\Pi}_{sd}) \leq\geq 0$. In the third situation, we have $UB - q_0(\hat{\Pi}_{sd}) < 0$ and $W^k - q_k(\hat{\Pi}_{sd}) < 0$. In all the cases, the following condition holds:

$$\min_{0 \leq t \leq p} \left\{ (\bar{q}_t + \delta_t) - q_t(\hat{\Pi}_{sd}) \right\} = -\alpha, with\ \alpha > 0. \tag{2.14}$$

When we evaluate the objective function of $P(\delta)$ in $\tilde{\Pi}_{sd}$, we have that $UB - q_0(\tilde{\Pi}_{sd}) \geq 0$ and $W^k - q_k(\tilde{\Pi}_{sd}) \geq 0$, $\forall k = 1, \ldots, p$.

It is evident that:

$$\min_{0 \leq t \leq p} \left\{ (\bar{q}_t + \delta_t) - q_t(\tilde{\Pi}_{sd}) \right\} = \beta, with\ \beta \geq 0. \tag{2.15}$$

From (2.14) and (2.15), we have that $-\alpha \leq \beta$.

This contradicts condition (2.13). □

Theorem 2.3.1 states that the search procedure, that makes use of the optimal cost of $P(\delta)$, does not identify a solution belonging is $SUB(\delta)$ only if $SUB(\delta) = \emptyset$. By the definition of this subregion, it is evident that if the solution of $P(\delta)$ does not belong to $SUB(\delta)$, then no more improvement on the cost is possible and the following corollary is derived.

Corollary 2.3.2. *Let* $\Pi(\delta)$ *be the optimal solution of* $P(\delta)$*. If* $\Pi(\delta) \notin SUB(\delta)$*, then* UB *is the optimal solution.*

Proof. If $\Pi(\delta) \notin SUB(\delta)$, then $SUB(\delta) = \emptyset$. This means that no solutions Π such that $c(\Pi) \leq UB$ exist, thus UB represents the optimal cost. □

The general scheme of the proposed Interactive Search Strategy Algorithm ($ISSA$, for short) is depicted in **Algorithm 1**.

Algorithm 1 $ISSA$

```
repeat
  Compute the new reference point defined by $\delta$.
  Determine the path $\Pi(\delta)$ solving the problem $P(\delta)$.
  if $c(\Pi(\delta)) < UB$ and $w^k(\Pi(\delta)) \leq W^k \forall\ k = 1, \ldots, p$ then
    set $UB = c(\Pi(\delta))$.
  end if
until $UB$ is modified
```

At each iteration of $ISSA$, a new reference point is computed. In the course of the algorithm, the search direction δ is defined using information related to the previous found feasible solution. It is important to point out that at the first iteration of $ISSA$, an initial feasible solution should be available. In addition, the lower the value of such a solution, the faster $ISSA$. In other words, a good UB allows of reducing greatly the solution space, thus $ISSA$ is more likely to perform fewer iterations.

Let π denote the number of possible paths from node s to a generic node $i \in N$, the following theoretical results holds.

Lemma 2.3.3. *In the worst case, the complexity of* $ISSA$ *is*

$$\mathcal{O}\left([UB - c(\Pi^*_{sd})]\pi \left|N\right|^2 p\right).$$

Proof. At each iteration, the operations executed by $ISSA$ refer to: the computation of a new reference point that takes $\mathcal{O}(1)$; the operations executed by the label correcting method to solve $P(\delta)$ that are bounded by $\mathcal{O}\left(\pi \left|N\right|^2 p\right)$; the **if** condition that takes $\mathcal{O}(p+1)$ and the operation executes in the **if** condition, that is $\mathcal{O}(1)$. Thus, the operations executed at each iteration of $ISSA$ are bounded by $\mathcal{O}\left(\pi \left|N\right|^2 p\right)$. This bound is related to the operations executed by the label correcting method proposed in [74] that is the same depicted in

Section 2.3.1 without the first **if** of **Step 2**. In the worst case, there is one arc connecting each pair of nodes. Thus, at each iteration, the second **if** of **Step 2** is invoked $\mathcal{O}(|N|)$ times and it takes $\mathcal{O}(p)$ operations. Anytime a node is selected, the **for** loop is invoked. In the worst case, a node is added to the list L π times. Since the **for** loop is invoked $\pi|N|$ times, the complexity is $\mathcal{O}\left(\pi|N|^2 p\right)$.

Refering to $ISSA$, in the worst case, the number of reference points to be considered is $[UB - c(\Pi^*_{sd})]$, that is all the Pareto solutions with different cost value have to be processed. Thus, the worst case complexity of the proposed method is $\mathcal{O}\left([UB - c(\Pi^*_{sd})]\pi|N|^2 p\right)$. □

In the next section, we introduce a label correcting method to obtain an initial UB, that is used to initialize $ISSA$. In addition, we introduce a formulation, based on the reference point methodology, that allows us to compute a lower bound (LB) for the $\mathcal{RCSPP}$. The proposed formulation has an useful property that allows us to check the optimality of the obtained solution.

2.3.1 Upper and Lower bounds

The strategy developed to compute the UB, can be viewed as a modified version of the label correcting method described in [74]. The aim is to find a feasible solution for problem (2.4), thus our modification takes into account constraints (2.2). In particular, if a partial solution does not satisfy constraints (2.2), then it is not taken into account. It is evident that until a feasible partial solution is not found, the procedure stores the partial solutions that are optimal if constraints (2.2) are relaxed.

Since the optimal solution is not formed by optimal subpaths, the proposed procedure allows to determine a suboptimal solution for the $\mathcal{RCSPP}$.

With the aim of improving the UB, we consider lower bound on the resources consumption. In other words, for each resource k, lb_j^k represents the least resource path from node j to node d. In this manner, we avoid to store partial solutions that cannot be part of the optimal solution.

It is assumed that a label y_j is associated with each node $j \in \mathcal{N}$ and a set L stores the nodes to be processed.

The modified version of the algorithm proposed in [74], used to find a suboptimal solution for the $\mathcal{RCSPP}$, is presented in **Algorithm 2**.

Algorithm 2 Label correcting method for computing initial UB

```
Step 0 (Initialization)
Set: $L = \{s\}$; $y_s = 0$, $y_i = +\infty, \forall i \in N, i \neq s$.

Step 1 (Node selection)
Select from $L$ a node $i$.

Step 2 (Node scan)
for all $j \in \mathcal{N} : (i,j) \in \mathcal{A}$ and $j \neq s$ do
  if $q_t(\Pi_{si}) + w_{ij}^t + lb_j^t \leq W^t$; $t = 1, \dots, p$ then
    if $y_j < \min_{0 \leq t \leq p}\{\bar{q}_t - [q_t(\Pi_{si}) + q_t^{(i,j)}]\}$ and $j \notin \Pi_{si}$ then
      Set: $y_j = \min_{0 \leq t \leq p}\{\bar{q}_t - [q_t(\Pi_{si}) + q_t^{(i,j)}]\}$,
      add $j$ to $L$ if $j$ does not already belong to it.
    end if
  end if
end for

Step 3 (Termination check)
if $L = \emptyset$ then
  STOP.
else
  Go to Step 1.
end if
```

As mentioned above, we also propose a strategy to find a LB for the $\mathcal{RCSPP}$. In particular, a LB for the $\mathcal{RCSPP}$ is determined by solving the following problem:

$$P(LB) \quad \max_{\Pi \in \bar{\Pi}} \min_{0 \le t \le p} \{\bar{q}_t^{LB} - q_t(\Pi)\}. \tag{2.16}$$

where $\bar{q}_t^{LB} = [c(\Pi_{sd}^c), W^1, \ldots, W^p]$.

It is important to point out that if the optimal path $\Pi(LB)$ of problem $P(LB)$ is feasible, then $\Pi(LB)$ is the optimal solution for the $\mathcal{RCSPP}$. This theoretical result is stated in Theorem 2.3.4 below.

Theorem 2.3.4. *If $\Pi(LB) \in \bar{\Pi}_0$, then $\Pi(LB)$ is optimal for the $\mathcal{RCSPP}$.*

Proof. Let Π_{sd}^* be the optimal solution for the $\mathcal{RCSSP}$. Let us consider a generic path $\Pi_{sd} \in \bar{\Pi}_0$. It is evident that $c(\Pi_{sd}^*) \le c(\Pi_{sd})$.

Evaluating (2.16) in Π_{sd} and in Π_{sd}^* leads to the following results:

$$\min_{0 \le t \le p} \{\bar{q}_t^{LB} - q_t(\Pi_{sd})\} = c(\Pi_{sd}^c) - c(\Pi_{sd}); \tag{2.17}$$

and

$$\min_{0 \le t \le p} \{\bar{q}_t^{LB} - q_t(\Pi_{sd}^*)\} = c(\Pi_{sd}^c) - c(\Pi_{sd}^*). \tag{2.18}$$

Since $c(\Pi_{sd}^*) \le c(\Pi_{sd})$, it follows that the maximum is achieved in Π_{sd}^*. This implies that:

$\min_{0 \le t \le p}\{\bar{q}_t^{LB} - q_t(\Pi_{sd}^*)\} > \min_{0 \le t \le p}\{\bar{q}_t^{LB} - q_t(\Pi)\}, \quad \forall \Pi \in \bar{\Pi}_0, \Pi \neq \Pi_{sd}^*$. □

2.4 Numerical experiments

The goal of this Section is to evaluate the numerical behaviour of the proposed algorithm. In addition, two solution approaches proposed in the scientific literature to solve the $\mathcal{RCSPP}$ have been considered. In particular, the best performing version of the Modified Label-Setting Algorithm ($MLSA$, for short) proposed in [60] and the solution approach based on the k-shortest paths algorithm (in the sequel refer to as kSP) defined in [125] have been compared with the proposed method. It is important to point out that both $MLSA$ and kSP have been defined to solve the $\mathcal{RCSPP}$ with at most one resource constraint. All the aforementioned algorithms have been implemented in java language and have been tested by using an Intel(R) Core(TM) i7 CPU M 620 PC, 2.67 GHz, RAM 4.00 GB, under Microsoft 7 operating system.

2.4.1 Test Problems

The test problems used to evaluate the behaviour of the proposed solution approach can be grouped in 3 different sets.

The first two sets refer to problems considered in [60]. In particular, the problems of the first set (referred to as D1) are those belonging to the Problem Class 4 of [60]. They are defined on networks with a grid structure, with randomly generated costs and weights. D1 contains four test problems, characterized by a different number of nodes (i.e., 10002, 40002, 70002, and 135002). The second set (referred to as D2) coincides with Problem Class 3 presented in [60]. These networks are based on graphs constructed from digital elevation models (DEMs). For each test problem, two different instances (i.e., class L and M) are generated depending on the value of the bound limit on the resource consumption. The characteristics of these problems are given in Table 2.1.

The third set of test problems (referred to as S) contains the instances considered in Santos et al. ([125]) and provided by the same authors. The characteristics of these problems are reported in Table 2.2. For each problem, ten instances are generated and for each

Test	Nodes	Arcs	Density (Arcs/Nodes)
D1_1	10002	29900	2.99
D1_2	40002	119800	2.99
D1_3	70002	209950	3.00
D1_4	135002	404850	3.00
D2_1	625	2400	3.84
D2_2	2005	9800	4.89
D2_3	5625	22200	3.95
D2_4	15625	62000	3.97
D2_5	22500	89400	3.97
D2_6	30625	121800	3.98
D2_7	40000	159200	3.98

Table 2.1: Characteristics of test problems belonging to sets D1 and D2

of them 5 different values for the bound limit are considered. Thus, the resulting 900 instances can be viewed as grouped in five different classes (group 1, ..., group 5), depending on the tightness of the resource constraint. More details about the set of test problems can be found in ([125]).

2.4.2 Computational results

In this Section, we evaluate the performances of $ISSA$ and we discuss about the quality of the upper and lower bounds determined by the strategies presented in Section 2.3.1. It is worth observing that the optimal solution for $P(LB)$ is determined by using the label correcting method proposed in [74]. A comparison with the state-of-art algorithms is also carried out.

Results on set S

First, let us consider the test problems belonging to the set S. The related results are reported in Tables 2.3 and 2.4, where for each test problem the computational times (in ms) and the number of iterations,

Test	Nodes	Arcs	Density (Arcs/Nodes)
S1	10000	15000	1.50
S2	10000	25000	2.50
S3	10000	50000	5.00
S4	10000	100000	10.00
S5	10000	150000	15.00
S6	10000	200000	20.00
S7	20000	30000	1.50
S8	20000	50000	2.50
S9	20000	100000	5.00
S10	20000	200000	10.00
S11	20000	300000	15.00
S12	20000	400000	20.00
S13	40000	60000	1.50
S14	40000	100000	2.50
S15	40000	200000	5.00
S16	40000	400000	10.00
S17	40000	600000	15.00
S18	40000	800000	20.00

Table 2.2: Characteristics of test problems belonging to set S

averaged on the related instances, are reported.

		$ISSA_{UB}$			kSP
	test	iter	time UB	time	time
group 1	S1	1.00	6.24	870.49	6022.80
	S2	1.00	3.12	1609.93	18802.00
	S3	1.00	4.68	4201.11	27412.30
	S4	1.00	37.44	7692.41	373402.20
	S5	1.00	893.89	10486.39	279524.40
	S6	1.00	1989.01	14618.85	14189012.60
	S7	1.00	1.56	3003.02	56803.30
	S8	1.00	4.68	5728.36	120598.50
	S9	1.00	15.60	13434.81	439588.50
	S10	1.00	173.16	28665.18	737883.90
	S11	1.00	1443.01	42218.55	1547797.00
	S12	1.00	8924.82	55927.92	4825502.90
	S13	1.00	15.60	13370.85	89941.30
	S14	1.00	15.60	22030.46	210953.40
	S15	1.00	65.52	53324.26	654895.90
	S16	1.00	595.40	112315.52	1654017.33
	S17	1.00	8021.57	159512.58	8042442.80
	S18	1.00	35516.75	212657.44	2440280.10
AVG		**1.00**	**3207.09**	**42314.90**	**1984160.07**
group 2	S1	1.00	3.12	814.33	12530.20
	S2	1.00	4.68	1564.69	18253.50
					continued on next page

continued from previous page

		$ISSA_{UB}$			kSP
	test	iter	time UB	time	time
	S3	1.00	31.20	3544.34	38308.40
	S4	1.00	366.60	6587.92	255040.30
	S5	1.00	3647.30	9820.26	511873.30
	S6	1.00	6647.20	13403.61	702564.30
	S7	1.00	3.12	3001.46	98912.40
	S8	1.00	6.24	5645.68	267519.60
	S9	1.00	76.44	12314.72	434444.00
	S10	1.00	2134.09	26546.69	1176277.90
	S11	1.00	11511.31	37964.40	1118095.00
	S12	1.00	26311.13	51099.69	2515935.70
	S13	1.00	17.16	12068.24	126038.10
	S14	1.00	42.12	22022.66	276217.00
	S15	1.00	636.48	48750.31	763809.70
	S16	1.00	10449.47	97981.63	3574890.50
	S17	1.00	54835.91	146503.66	1761081.20
	S18	1.00	118368.88	203401.90	4652507.80
AVG		**1.00**	**13060.69**	**39057.57**	**1016905.49**
group 3	S1	1.00	7.80	800.29	9612.30
	S2	1.00	7.80	1580.29	14855.10
	S3	1.00	304.20	3464.78	56039.90
	S4	1.00	2333.78	6475.60	252112.00
	S5	1.00	7143.29	10350.67	2147393.20
	S6	1.00	10918.51	14338.05	405596.00
	S7	1.00	3.12	2998.34	143212.30
	S8	1.00	32.76	5664.40	300948.80
	S9	1.00	1265.17	12461.36	352407.70
	S10	1.00	8567.57	25217.56	1490010.30
	S11	1.00	25930.49	39560.29	812205.50
	S12	1.00	42182.67	53179.18	2765203.20
	S13	1.00	29.64	12467.60	170866.30
	S14	1.00	435.24	22584.26	318891.30
	S15	1.00	8338.25	48917.23	530171.50
	S16	1.00	47382.70	94957.81	1284840.67
	S17	1.00	111197.51	159156.90	1411866.60
	S18	1.00	190516.22	230248.12	3113543.50
AVG		**1.00**	**25366.48**	**41356.82**	**865543.12**
group 4	S1	1.00	3.12	801.85	15498.30
	S2	1.00	37.44	1627.09	16233.70
	S3	1.00	893.89	3616.10	63905.60
	S4	1.00	4115.31	7055.93	201423.60
	S5	1.00	9389.70	11709.44	1768348.70
	S6	1.00	13734.33	16314.58	711878.90
	S7	1.00	9.36	2964.02	210424.90
	S8	1.00	109.20	5812.60	191087.10
	S9	1.00	3778.34	13141.52	492112.40
	S10	1.00	15082.18	27835.26	768624.80
	S11	1.00	34533.94	43647.52	941196.10
	S12	1.00	52038.81	59951.18	4185858.10
	S13	1.00	45.24	12756.20	234069.30
	S14	1.00	1580.29	23682.51	232858.60
	S15	1.00	20379.97	53882.75	598947.20
	S16	1.00	74116.08	106109.28	1445965.17
	S17	1.00	147461.51	183839.38	2599483.80
	S18	1.00	238230.69	266471.55	6539820.70
AVG		**1.00**	**34196.63**	**46734.38**	**1178763.16**
group 5	S1	1.00	7.80	829.93	22078.00
	S2	1.00	115.44	1726.93	16572.40
	S3	1.00	1494.49	3820.46	16911741.30
	S4	1.00	5367.99	7689.29	128524.40
	S5	1.00	11193.07	13294.41	332041.30
	S6	1.00	15821.62	17997.84	478472.30
	S7	1.00	10.92	2962.46	847035.10
	S8	1.00	319.80	6007.60	195547.40
	S9	1.00	6332.08	13891.89	447496.70
	S10	1.00	20476.69	29632.39	857715.50
	S11	1.00	41187.38	48212.11	2771501.10
	S12	1.00	60721.83	66840.19	2017669.10
	S13	1.00	76.44	12909.08	880845.00
	S14	1.00	2920.34	24538.96	283554.40
	S15	1.00	30552.80	57612.73	584054.10
	S16	1.00	97695.63	117928.96	1236643.83
	S17	1.00	201572.01	233886.06	1641821.10

continued on next page

continued from previous page

		$ISSA_{UB}$			kSP
	test	iter	time UB	time	time
	S18	1.00	271192.14	296419.06	3702571.20
AVG		**1.00**	**42614.36**	**53122.24**	**1853104.68**

Table 2.3: For groups 1, 2, 3, 4 and 5 we report the results obtained with $ISSA$ and kSP. In the column **iter** the number of iterations executed by $ISSA$, averaged on ten instances for each test problem, are reported. Column **time UB** gives the execution time needed to obtain the UB. The average execution times are reported in column **time**.

		$ISSA_{noUB}$		kSP
	test	iter	time	time
group 1	S1	1.00	811.21	6022.80
	S2	1.40	1486.69	18802.00
	S3	1.60	5981.08	27412.30
	S4	2.50	20252.05	373402.20
	S5	3.70	46895.46	279524.40
	S6	3.50	54743.87	14189012.60
	S7	1.00	2697.26	56803.30
	S8	1.40	7508.33	120598.50
	S9	1.80	22925.91	439588.50
	S10	2.10	59958.98	737883.90
	S11	2.20	99088.72	1547797.00
	S12	3.00	170424.85	4825502.90
	S13	1.10	12926.24	89941.30
	S14	1.20	26624.69	210953.40
	S15	2.30	137787.88	654895.90
	S16	2.00	229313.67	1654017.33
	S17	4.10	704360.12	8042442.80
	S18	3.50	817047.44	2440280.10
AVG		**2.19**	**134490.80**	**1984160.07**
group 2	S1	1.10	837.73	12530.20
	S2	1.50	2127.85	18253.50
	S3	1.70	5642.56	38308.40
	S4	2.70	18807.48	255040.30
	S5	4.70	48522.55	511873.30
	S6	5.30	74116.07	702564.30
	S7	1.10	2648.90	98912.40
	S8	1.50	7915.49	267519.60
	S9	2.30	27864.90	434444.00
	S10	2.70	70967.97	1176277.90
	S11	3.60	146786.02	1118095.00
	S12	4.10	223766.27	2515935.70
	S13	1.30	15366.10	126038.10
	S14	1.50	31415.48	276217.00
	S15	2.60	129990.95	763809.70
	S16	3.17	326962.50	3574890.50
	S17	4.20	683978.58	1761081.20
	S18	4.80	1065415.07	4652507.80
AVG		**2.77**	**160174.03**	**1016905.49**
group 3	S1	1.60	1291.69	9612.30
	S2	2.10	2712.86	14855.10
	S3	3.40	9648.66	56039.90
	S4	4.30	25638.76	252112.00
	S5	4.60	46546.02	2147393.20
	S6	4.10	74371.92	405596.00
	S7	1.10	2772.14	143212.30
	S8	1.90	10356.91	300948.80
	S9	2.90	33368.61	352407.70
	S10	3.80	84131.34	1490010.30
	S11	4.20	150166.56	812205.50
	S12	4.10	222454.31	2765203.20
	S13	1.70	15664.06	170866.30

continued on next page

continued from previous page				
		$ISSA_{noUB}$		kSP
	test	iter	time	time
	S14	2.00	34487.14	318891.30
	S15	3.10	149530.08	530171.50
	S16	4.83	433997.38	1284840.67
	S17	4.30	771000.62	1411866.60
	S18	3.90	1084869.95	3113543.50
AVG		**3.22**	**175167.17**	**865543.12**
group 4	S1	1.60	1182.49	15498.30
	S2	2.10	2971.82	16233.70
	S3	3.40	10682.95	63905.60
	S4	4.30	27412.50	201423.60
	S5	4.60	48032.71	1768348.70
	S6	4.10	59359.94	711878.90
	S7	1.10	2903.18	210424.90
	S8	1.90	10057.38	191087.10
	S9	2.90	35095.55	492112.40
	S10	3.80	95132.53	768624.80
	S11	4.20	165761.98	941196.10
	S12	4.10	216286.03	4185858.10
	S13	1.70	19325.40	234069.30
	S14	2.00	41018.90	232858.60
	S15	3.10	149648.64	598947.20
	S16	4.83	481741.49	1445965.17
	S17	4.30	710601.72	2599483.80
	S18	3.90	942938.68	6539820.70
AVG		**3.22**	**167786.33**	**1178763.16**
group 5	S1	1.60	1218.37	22078.00
	S2	2.30	3279.14	16572.40
	S3	3.22	10344.60	16911741.30
	S4	4.00	27089.57	128524.40
	S5	4.30	49065.43	332041.30
	S6	3.80	60455.07	478472.30
	S7	1.10	2953.10	847035.10
	S8	2.10	11757.80	195547.40
	S9	3.20	40844.18	447496.70
	S10	3.70	98489.67	857715.50
	S11	3.60	158431.50	2771501.10
	S12	3.80	221405.98	2017669.10
	S13	1.80	20404.93	880845.00
	S14	2.00	41633.55	283554.40
	S15	3.00	156231.88	584054.10
	S16	3.67	365528.54	1236643.83
	S17	3.50	617997.96	1641821.10
	S18	3.90	1001420.34	3702571.20
AVG		**3.03**	**160475.09**	**1853104.68**

Table 2.4: For each group we report the results obtained with $ISSA$ and kSP. In the column **iter** the number of iterations executed by $ISSA$, averaged on ten instances for each test problem, are reported. The average execution times are given in column **time**.

	gap LB - Opt					**No. Opt LB**					**time LB**				
test	**1**	**2**	**3**	**4**	**5**	**1**	**2**	**3**	**4**	**5**	**1**	**2**	**3**	**4**	**5**
S1	7.87%	7.22%	4.49%	4.77%	3.73%	2	2	6	6	6	808.09	742.56	787.81	775.32	775.33
S2	19.42%	11.10%	4.95%	5.90%	5.29%	1	1	4	4	6	1471.09	1416.49	1483.57	1517.89	1494.49
S3	30.02%	26.22%	17.41%	12.55%	11.51%	0	0	3	4	6	3109.10	2939.06	3183.98	3235.46	3451.09
S4	32.61%	26.40%	12.88%	9.73%	2.06%	0	1	1	3	6	6386.68	5814.16	6406.96	6795.40	7300.85
S5	37.61%	34.00%	13.19%	6.19%	15.83%	0	0	3	4	7	9643.98	9258.66	10659.55	11466.07	12905.96
S6	45.68%	30.24%	17.19%	8.95%	3.00%	0	0	3	5	6	12408.32	12618.92	14787.34	15639.10	17710.79
S7	5.24%	5.24%	5.24%	4.38%	5.42%	4	4	4	4	4	2692.58	2667.62	2836.10	2703.50	2832.98
S8	7.72%	7.65%	5.92%	1.81%	3.16%	5	5	6	6	6	4977.99	5070.03	5723.68	5416.35	5801.68
S9	23.99%	12.60%	5.71%	4.75%	3.37%	1	3	5	6	7	10728.19	10960.63	12445.76	12367.76	13584.57
S10	34.04%	26.28%	14.57%	10.01%	10.56%	0	0	3	5	6	22635.75	22835.43	24390.76	26215.97	29044.27
S11	38.44%	27.93%	14.04%	9.03%	9.44%	0	0	4	6	7	33454.41	34811.62	37423.08	41806.71	47642.71
S12	38.80%	28.90%	10.78%	0.89%	2.08%	0	0	5	6	6	46443.06	48811.15	51377.37	57456.73	64639.01
S13	7.83%	5.81%	3.86%	2.07%	3.36%	3	5	5	6	6	11743.76	11971.52	11431.75	11576.83	11556.55
S14	9.60%	8.00%	4.27%	4.87%	3.96%	2	3	5	6	6	21626.42	21336.26	21127.22	21757.46	22161.50
S15	29.04%	19.80%	7.64%	4.79%	4.40%	0	0	5	6	6	46859.58	44622.53	47377.50	53693.98	58257.01
S16	28.51%	23.77%	10.12%	0.53%	1.89%	1	1	3	6	6	85322.15	85462.55	95327.01	107807.09	113436.13
S17	36.39%	25.11%	9.49%	6.46%	3.27%	0	0	4	6	9	135722.43	144218.24	166424.99	179518.15	199912.16
S18	21.63%	11.14%	4.23%	1.67%	5.75%	0	0	4	7	9	194651.81	207567.13	231711.41	267282.75	295225.65
AVG	**25.25%**	**18.75%**	**9.22%**	**5.52%**	**5.45%**	**1.06**	**1.39**	**4.06**	**5.33**	**6.39**	**36149.19**	**37395.81**	**41383.66**	**45946.25**	**50429.60**

Table 2.5: For each test problem we report the results averaged on ten instances. The column **gap LB-Opt** gives the average percentage LB gap. The column **No. Opt LB** indicates the number of instances for each test problem for which the obtained LB is the optimal solution. The average computational costs are reported in column **time LB**. Each group refers to a different value of the resource limit. The higher the number of the group. the higher the value of the resource limit.

Two different versions of $ISSA$ have been tested: the former (referred to as $ISSA_{UB}$) uses the UB, computed as described in Section 2.3.1, in the latter ($ISSA_{noUB}$, for short) the UB is initialized with the cost of the least resource path.

When analyzing results reported in Table 2.3, one first observes that $ISSA_{UB}$ executes exactly one iteration. This means that the solutions found by the procedure used to obtain the initial upper bound are optimal. In addition, it is evident that the average computational time, on test problems S1-S18 increases when the bound limit on the resource consumption increases. This behaviour can be justified by observing that when an high resource limit is considered, the fathoming rule that uses lower bounds on the resource consumption allows to label a higher number of nodes than in the case in which the resource constraint is tight. As a consequence, the higher the resource limit, the higher the average time to solve the test problems, that is the time needed to find the UB plus the execution time of $ISSA$. In particular, it is observed that the average computational time on the test problems of group 1 is equal to 45521.99 and increases of 14%, 47%, 78% and 110% for groups 2, 3, 4 and 5, respectively.

As expected, $ISSA_{noUB}$ executes a number of iterations greater than one. In this case, the average computational cost is equal to 134490.80, 160174.03, 175167.17, 167786.33 and 160475.09 for groups 1, 2, 3, 4 and 5, respectively (see Table 2.4). The computational time increases of 10%, 30%, 25% and 19% respect to the average execution time obtained when the test problems of group 1 are solved, for groups 2, 3, 4 and 5, respectively. This behaviour can be justified by considering the number of iterations executed by $ISSA_{noUB}$. In particular, the increase in the average number of iterations performed, when the test problems of group 1 are solved, is equal to 26%, 47%, 47% and 38% for groups 2, 3, 4 and 5, respectively.

By comparing the results collected in Tables 2.3 and 2.4, it is evident that $ISSA_{UB}$ is faster than $ISSA_{noUB}$. In particular, $ISSA_{UB}$

is on average 2.34 times faster than $ISSA_{noUB}$. This behaviour is observed for all considered groups.

As far as the comparison with the method proposed in [125] is concerned, $ISSA_{UB}$ is 20.22 times faster than kSP. It is important to point out that kSP performs also worse than $ISSA_{noUB}$. In particular, $ISSA_{noUB}$ is 8.64 times faster than kSP.

The problem $P(LB)$ defined in Section 2.3.1 allows us to find good lower bounds, for the test problems of set S. The higher the resource limit, the lower the distance from the optimal solution. In particular, the average optimality gap is 25.25%, 18.75%, 9.22%, 5.52% and 5.45% for groups 1, 2, 3, 4 and 5, respectively (see Table 2.5). This behaviour can be justified by taking into account the number of instances that are optimality solved for each test problem. Indeed, the average number of times in which the LB coincides with the optimal solution is 1.06, 1.39, 4.06, 5.33 and 6.39 for groups 1, 2, 3, 4 and 5, respectively. From Table 2.5, it is evident that, when a higher value of the resource limit is considered, a label correcting method used to solve the problem $P(LB)$ is more likely to find the optimal solution. This specific behaviour can be explained by observing that since the objective function of $P(LB)$ is of ASF type, solving the problem allows to find a properly Pareto optimal solution that is nearest to the reference point. The reference point defined for $P(LB)$ has in the cost component the minimum cost. When the other components have a value too far from the resources value of the least cost path, the solution found by the label correcting method presents resource consumptions far as well from the resource components of the reference point. Instead, when the resource components of the reference point have an high value, the resources consumption of the properly Pareto solution is closer to the resource components of the reference point. Since these values are set equal to the resource limits, thus the solution find by the label correcting method is closer to the feasible region.

In addition, the computational time increases when the resource limit increases. This is justified by the fact that the resource components of the reference point increase as well. This means that a path with an higher value of the resources consumption should be found. It is evident that, since the resource consumption on the arcs is a vector of positive values, a longer path is found for higher values of the references related to the resources.

Results on sets D1 and D2

A behaviour, similar to that underlined for the test problems of set S, has been observed for sets D1 and D2. In particular, as shown in Table 2.6 the computational time to find the LB for the L-instances of set D2 is equal to 20495.00, while for the M-instances the average execution time is equal to 121182.69. In addition, the average optimality gap is equal to 33.80% and 6.13% for L-instances and M-instances, respectively. It is important to point out that the optimal solution of $P(LB)$ is also optimal for the related $\mathcal{RCSPP}$ in 1 out of 26 instances of L type, instead, for the M-instances the LB is the optimal solution for 6 out of 27 instances. This behaviour can be explained similarly to what has been reported for the set S.

When we consider the sets D1, it is evident that the average optimality gap is better than that obtained for the problems of set D2 and S, even though no test problems are optimality solved (see Table 2.6). In particular, the LB is far from the optimal solution of about 3.84% and 0.54% for the L-instances and M-instances, respectively. Also in this case, the higher the resource limit, the better the LB's quality.

For problems of sets D1 and D2, an UB can be computed by the preprocessing procedure, defined in [60]. We have used both this upper bound and the one obtained with the procedure defined in Section 2.3.1. We refer to $ISSA$ with the UB obtained from the preprocessing as $ISSA_{PrepUB}$. Our computational results suggest that $ISSA_{UB}$

is faster than $ISSA_{PrepUB}$ (see Table 2.7). In particular, the former is 2.15 and 3.28 times faster than the latter for the L-instances and the M-instances belonging to set D2, respectively. Instead, for the instances in the set D1, $ISSA_{UB}$ is 2.42 and 1.02 times faster than $ISSA_{PrepUB}$ for L-instances and M-instances, respectively. This performance can be explained by analyzing the results collected in Table 2.6. Indeed, for the instances of set D2 we can observe an improvement of our UB respect to that obtained with the preprocessing procedure equal to the 16.90% and 57.61% for L-instances and M-instances, respectively. As a consequence, the number of iterations of $ISSA_{UB}$ is less that the one executed by $ISSA_{PrepUB}$. In particular, as shown in Table 2.7, the number of iterations executed by $ISSA_{UB}$ and $ISSA_{PrepUB}$ is on average equal to 1.93 and 7.98, respectively.

The same behavior has been observed for the instances of the set D1. In particular, the average number of iterations executed by $ISSA_{UB}$ is equal to 4.38, instead the average number of iterations performed by $ISSA_{PrepUB}$ is equal to 8.25 (see Table 2.7). The improvements on the number of iterations is less evident than that obtained for the instances belonging to the set D2. As a matter of fact, the average improvement of our UB respect to the upper bound obtained with the preprocessing is of the 1.62% and 6.47% for the L-instances and M-instances, respectivaly. In addition, as underlined for the problems in the set S, the higher the resource limit, the higher the average computational time (see Tables 2.6 and 2.7). This is true for both sets D1 and D2.

When comparing the computational results of $ISSA$ with those obtained by $MLSA$ (see Table 2.7) it is evident that the proposed solution approach outperforms $MLSA$. In particular, the best version of $ISSA$, that is $ISSA_{UB}$, is on average 14.30 and 32.33 times faster than $MLSA$ for problems belonging to the set D1 and D2, respectively. In addition, the proposed method is faster than $MLSA$ even though we use the upper bound obtained with the preprocessing. In particular,

$ISSA_{PrepUB}$ is on average 13.88 and 11.99 times faster than $MLSA$ for problems in the sets D1 and D2, respectively. It is important to point out that $ISSA$ shows worse performance than $MLSA$ only for the L-instances of D1, in particular, $MLSA$ is 2.29 times faster than $ISSA_{UB}$ (see Table 2.7).

2.4.3 Final remarks

The results collected from our computational studies can be summarize as follows:

1) the proposed method, to obtain an upper bound for the $\mathcal{RCSPP}$, gives a solution that improves the upper bound provided by the preprocessing procedure ([60]), on the instances of sets D1 and D2. For all the problems belonging to the set S, the proposed method provides the optimal solution;

2) the model defined to find a lower bound allows us to obtain the optimal solution for several instances of set S. The less thight the resource constraints, the lower the optimality gap, this behaviour is observed for all considered test problems. The best values of lower bound are obtained for the instances belonging to the set D1;

3) the proposed solution approach to optimality solve the $\mathcal{RCSPP}$ outperforms the best known algorithms. The proposed procedure behaves the best either when it is initialized with our upper bound or with a worst feasible solution. An exception is observed only for the L-instances of set D1, for which $MLSA$ behaves slight better.

2.5 Conclusions and future work

In this paper, we have addressed the resource constrained shortest path problem. A new solution approach, based on the reference point methodology, has been defined. In addition, using the concept of

Test		% improving UB	time UB	gap LB - Opt	time LB	No. Opt LB
L-instances	D1_1	1.14%	15.60	3.39%	280.80	0
	D1_2	1.65%	3681.62	3.89%	144004.52	0
	D1_3	1.37%	3619.22	4.12%	177653.94	0
	D1_4	2.31%	21886.94	3.95%	1347302.64	0
AVG		**1.62%**	7300.85	**3.84%**	**417310.48**	0/4
M-instances	D1_1	6.36%	3603.62	0.60%	7534.85	0
	D1_2	6.41%	96923.42	0.58%	192817.24	0
	D1_3	6.54%	377974.82	0.60%	607218.29	0
	D1_4	6.58%	1849891.06	0.54%	3270950.97	0
AVG		**6.47%**	582098.23	**0.58%**	**1019630.34**	0/4
L-instances	D2_1	9.62%	0.00	4.22%	7.80	1
	D2_2	16.90%	15.60	5.47%	26.00	0
	D2_3	15.13%	42.90	29.06%	210.60	0
	D2_4	21.12%	331.50	36.97%	3989.73	0
	D2_5	18.04%	717.60	50.85%	17378.51	0
	D2_6	17.16%	1014.01	50.69%	18400.32	0
	D2_7	20.33%	2086.51	59.36%	103452.06	0
AVG		**16.90%**	601.16	**33.80%**	**20495.00**	1/26
M-instances	D2_1	51.96%	5.20	12.43%	26.00	2
	D2_2	50.59%	182.00	0.26%	384.80	2
	D2_3	39.90%	618.80	6.27%	2064.41	0
	D2_4	53.29%	9297.66	3.40%	23467.75	1
	D2_5	67.94%	36597.84	8.09%	85249.35	1
	D2_6	66.57%	63258.41	6.80%	193774.04	0
	D2_7	73.03%	243911.46	5.64%	543312.48	0
AVG		**57.61%**	50553.05	**6.13%**	**121182.69**	6/27

Table 2.6: The percentage improvement of our UB respect to that obtained with the preprocessing is given in column **% improving UB** . In column **time UB** the execution time to obtain the UB is reported. The column **gap LB-Opt** gives the average farness of the LB from the optimal solution. The average computational costs are reported in column **time LB**. The column **No. Opt LB** indicates the number of instances for each test problem for which the obtained LB is the optimal solution.

Chebyshev distance, models and methods for finding upper and lower bounds have been developed. The proposed methods have been tested on benchmark test problems taken from the literature. Extensive computational tests have been carried out to evaluate the behaviours of the proposed solution approaches. In addition, a comparison with the best known label setting algorithm ([60]) and the best known k-shortest paths based approach ([125]) is provided.

From our computational studies, we can conclude that the proposed optimal solution strategy is competitive with the state-of-art approaches. In addition, the models and methods proposed for finding upper and lower bounds allows us to obtain satisfactory results.

It is important to point out that very recently an efficient preprocessing procedure for the $\mathcal{RCSPP}$ has been defined by Boland and Muhandiramge [111]. A comparison of the quality of our upper bound

		$ISSA_{UB}$			$ISSA_{PrepUB}$		**MLSA**
Test		**iter**	**time UB**	**time**	**iter**	**time**	**time**
L-instances	D1_1	3.00	15.60	2496.02	5.00	4383.63	2028.01
	D1_2	7.00	3681.62	2324586.50	11.00	3794833.53	943884.05
	D1_3	3.00	3619.22	1330735.33	7.00	3246427.61	975084.25
	D1_4	6.00	21886.94	21926065.35	14.00	55012988.25	9260406.56
AVG		**4.75**	**7300.85**	**6395970.80**	**9.25**	**15514658.25**	**2795350.72**
M-instances	D1_1	2.00	3603.62	11310.07	7.00	38547.85	426179.13
	D1_2	4.00	96923.42	564333.62	8.00	1190911.63	38332066.52
	D1_3	3.00	377974.82	1180053.96	7.00	2736491.54	135054262.13
	D1_4	7.00	1849891.06	15635169.02	7.00	16179940.12	381672818.76
AVG		**4.00**	**582098.23**	**4347716.67**	**7.25**	**5036472.78**	**138871331.63**
L-instances	D2_1	1.00	0.00	0.00	1.50	7.80	10.40
	D2_2	1.00	15.60	36.40	2.67	62.40	379.60
	D2_3	1.25	42.90	167.70	3.25	588.90	3662.12
	D2_4	1.25	331.50	1357.21	3.25	3377.42	76927.99
	D2_5	1.00	717.60	2956.22	1.50	4680.03	189810.32
	D2_6	4.00	1014.01	10159.57	12.00	52131.63	324762.88
	D2_7	6.25	2086.51	52603.54	10.00	92602.19	1299893.93
AVG		**2.25**	**601.16**	**9611.52**	**4.88**	**21921.48**	**270778.18**
M-instances	D2_1	1.67	5.20	41.60	8.67	161.20	192.40
	D2_2	1.00	182.00	270.40	4.00	759.20	9204.06
	D2_3	2.33	618.80	4648.83	4.33	5168.83	78161.70
	D2_4	1.00	9297.66	16270.90	10.00	152558.58	2173858.34
	D2_5	1.33	36597.84	89456.17	13.00	288617.45	10772144.65
	D2_6	3.00	63258.41	475636.65	17.00	964398.18	20879745.84
	D2_7	1.00	243911.46	487752.73	20.50	3274936.79	20549013.32
AVG		**1.62**	**50553.05**	**153439.61**	**11.07**	**669514.32**	**7780331.47**

Table 2.7: The column **iter** gives the iteration executed by $ISSA$. The average execution times are reported in column **time**. Under column **Time UB**, we give the computational cost to obtain the UB.

and that obtained with the preprocessing strategy of [111] will be the subject of future investigation.

Chapter 3

Dynamic programming approaches to solve the shortest path problem with forbidden paths [1]

Luigi Di Puglia Pugliese

Department of Electronics, Informatics and Systems, University of Calabria, 87036, Rende (CS), Italy

Francesca Guerriero

Department of Electronics, Informatics and Systems, University of Calabria, 87036, Rende (CS), Italy

Abstract

In this paper, the shortest path problem with forbidden paths is addressed. The problem under consideration is formulated as a particular instance of the resource constrained shortest path problem. Different versions of a dynamic programming based solution approach are defined and implemented. The proposed algorithm can be viewed as

[1]Submitted for pubblication in the journal *Optimization Methods & Software*.

an extension of the node selection approach proposed by Desrochers in 1988. An extensive computational test, on a meaningful number of random instances, is carried out with the purpose to assess the behaviour of the developed solution approaches. A comparison with the state-of-art method proposed to address the problem under study is also made. The computational results are very encouraging and highlight that the proposed algorithms turn out to be very efficient.

Keywords: shortest path problem, forbidden paths, dynamic programming.

3.1 Introduction

The Shortest Path Problem ($\mathcal{SPP}$, for short) is one of the most studied problems in network optimization ([46]; [57]; [65]). In its basic formulation, the objective is to determine the minimum cost path through a network from a given origin node to a destination node. Different polynomial time solution approaches have been developed in the scientific literature to address the $\mathcal{SPP}$ (e.g., see [47]; [57]; [99]). It is worth observing that various instances of the classical $\mathcal{SPP}$ have been formulated and studied. An important variant of the $\mathcal{SPP}$ is the constrained shortest path problem ($c\mathcal{SPP}$, for short), that has received great attention due to its practical importance. Indeed, many real situations can be formulated as a $c\mathcal{SPP}$ and, thus, various "constrained" versions of the basic $\mathcal{SPP}$ can be defined. For example, the path should include specific nodes, or a specific number of nodes (e.g., see [49]), or nodes within a pre-specified "covering" distance from each node in the network ([44]; [45]).

In general, the $c\mathcal{SPP}$ refers to the shortest path problem with one additional constraint, that establishes an upper limit on the sum of some other arc parameter (e.g., travel time) for the path ([81]). While the $\mathcal{SPP}$ is easy to solve (i.e., it has a polynomial complexity bound), the addition of this constraint generally results in problems belonging to the NP-hard class (e.g., see [71]). The problem of determining the minimum cost path with a limited resource constraint has been addressed by Dumitrescu and Boland in [60], where labelling and scaling algorithms are proposed. In order to improve the performances of the defined solution approaches, a preprocessing procedure is also defined. Recently, an improved approach to address the $c\mathcal{SPP}$, that is based on the solution of the k-shortest path problem ($k - \mathcal{SPP}$) has been proposed by Santos et al. in [125]. The authors introduce a new search direction that allows the outperformance of the state-of-art algorithms based on the $k - \mathcal{SPP}$.

The $c\mathcal{SPP}$ in which more than one constraint is included is referred to as shortest path problem with resource constraints ($\mathcal{SPPRC}$, for short). The dynamic programming framework has been used to develop efficient solution approaches for the $\mathcal{SPPRC}$. In particular, a label, representing the consumption of resources, is associated with each possible partial path. Dominance rules are used to fathom unpromising labels. In label correcting approaches, nodes are repeatedly treated and their labels extended. This solution strategy has been originally proposed by Desrochers ([50]) and it can be viewed as an extension of the Bellman-Ford algorithm, taking resource constraints into account.

It is worth observing that a label can be treated instead of a node. In this case, the selected label is used to generate a new label for each successor node. Dynamic programming approaches have also been developed by Desrochers and Soumis ([52]) and by Jaumard et al. ([92]).

Addressing the $\mathcal{SPPRC}$ by using a dynamic programming approach is closely related to deal with the multi-objective shortest path problems, which have received the attention of many researchers. In these problems the aim is to generate non-dominated paths (i.e., Pareto optimal paths). The methods developed in the literature usually concern multi-objective shortest path problems on graphs with non-negative lengths (see, i.e., [83] and [142]).

Another important class of solution approaches for the $\mathcal{SPPRC}$ uses Lagrangian relaxation technique, by taking advantage of the effectiveness of algorithms that solve the unconstrained version of the problem ([16]; [23]; [81]; [105]). A detailed survey devoted to the $\mathcal{SPPRC}$ is provided in [150].

In this paper, we address the shortest paths problem with forbidden paths (in the sequel referred to as $\mathcal{SPPFP}$), which has been introduced quite recently by Villeneuve and Desaulniers ([138]). The main aim is to find the shortest paths from an origin node to all other nodes of a directed graph, such that no paths in the optimal solution contain paths belonging to a given set (i.e., forbidden paths). The $\mathcal{SPPFP}$ arises in all the contexts in which a $\mathcal{SPP}$ must be modified, to exclude a set of paths, that do not satisfy a given feasibility criterion ([138]).

The first method appeared in the scientific literature that handles the $\mathcal{SPPFP}$, has been proposed in ([138]) and it has been defined by combining solution techniques developed for the $k - \mathcal{SPP}$ ([100]) together with a method defined to solve the keyword matching problem ([2]). The main idea is to construct an enlarged graph, such that solving the shortest path problem in this graph ensures that the optimal solution does not contain any forbidden path. The approach of Villeneuve and Desaulniers ([138]) is a polynomial time algorithm where the length of input is the size of the network and the number of arcs in the forbidden paths.

Recently, an improved version of the solution approach proposed in [138] has been presented in [30]. The improvement has been obtained by reducing the number of node of the enlarged graph insuring the correctness of the method. The modified solution approach of Villeneuve and Desaulniers ([138]) proposed in [30] is still a polynomial time algorithm.

A particular instance of the $\mathcal{SPPFP}$ has been addressed in [78], where a shortest path problem, in a road network with turn prohibition, is considered. In this case, the constraints are represented as a set of forbidden paths, each of them containing only two arcs; in the node shared by these arcs the turn is prohibited. To the best of our knowledge no papers have been published thus far with a comprehensive study of the practical behaviour of solution approaches for the $\mathcal{SPPFP}$.

In this work, we formulate the $\mathcal{SPPFP}$ as a variant of the $\mathcal{SPPRC}$, in which the constraints are represented by a set of paths, that cannot be included in the path solution. It is important to point out that, since the $\mathcal{SPPRC}$ belongs to the NP-hard class, each algorithm defined to solve the $\mathcal{SPPFP}$ as a specific instance of the $\mathcal{SPPRC}$ is inferior to the graph modification approach of Villeneuve and Desaulniers ([138]) and the improved version ([30]) from a worst case point of view. Indeed, the number of states grows exponentially with the number of forbidden paths. However, as it will be shown in the sequel, the suggested approaches outperform the original method of Villeneuve and Desaulniers ([138]) and its improvement ([30]) in practice.

The main contribution of this paper is twofold. On the one hand, we propose solution approaches for the $\mathcal{SPPFP}$, that in contrast to the state-of-art algorithms, work on the original graph. On the other hand, we empirically evaluate, for the first time, solution approaches for the $\mathcal{SPPFP}$.

The rest of the paper is organized as follows. In Section 2, we formulate the problem under consideration as a specific instance of the $\mathcal{SPPRC}$. The proposed solution approaches, based on dynamic programming optimization, are described in Section 3. Section 4 is devoted to the presentation of the computational results, obtained on the basis of an extensive testing phase. Finally, concluding remarks are given in Section 5.

3.2 Problem definition

The $\mathcal{SPPFP}$ is defined on a directed graph $\mathcal{D} = (\mathcal{N}, \mathcal{A})$, where $\mathcal{N} = \{1, \ldots, n\}$ denotes the set of nodes, and $\mathcal{A} \subseteq \mathcal{N} \times \mathcal{N}$ the set of arcs. It is assumed that the directed graph $\mathcal{D}$ is *simple*, where one arc at most connects any two nodes. A cost c_{ij} is associated with each arc $(i, j) \in \mathcal{A}$. A set of forward successor nodes $FS(i) = \{j|(i, j) \in \mathcal{A}\}$ is associated with each node $i \in \mathcal{N}$.

Given two distinct nodes o (referred to as origin node) and d (referred to as destination node), a path from o to d is a sequence of nodes $\Pi_{od} = \{o = i_1, \ldots, i_l = d\}, l \geq 2$ and a corresponding sequence of $l - 1$ arcs such that the $r - th$ arc in the sequence is $(i_r, i_{r+1}) \in \mathcal{A}$ for $r = 1, \ldots, l - 1$. Thus, each path contains at least one arc. A path is said to be *elementary*, if it does not contain repeated nodes, whereas it is said to be *simple* if it does not contain repeated arcs. It is worth observing that a simple path Π_{oj} (containing no repeated arcs) can be a non elementary path (i.e., a node can appear more than once). An *oriented cycle* is an elementary path for which the origin and destination nodes are the same.

The cost $z(\Pi_{od})$ of a path $\Pi_{od} = \{o = i_1, \ldots, i_l = d\}$ is defined as the sum of the costs of its arcs, that is $z(\Pi_{od}) = \sum_{(i,j)\in\Pi_{od}} c_{ij}$. In what follows, we assume that every cycle in $\mathcal{D}$ has a non-negative cost.

Let $\mathcal{F}$ denote the finite set of forbidden paths in $\mathcal{D}$. No assumptions are made about the characteristics of these paths. In what follows, $\dot{f}_\gamma$ and $\ddot{f}_\gamma$ denote the initial arc and the final arc of the forbidden path $f_\gamma, \gamma = 1, \ldots, |\mathcal{F}|$, respectively. Given an arc $a_k = (i, j) \in \mathcal{A}$ we denote with $\underline{a_k}$ the extreme node that is $\underline{a_k} = j$ and $\forall \gamma = 1, \ldots, |\mathcal{F}|$ we define the set $N^{(\gamma)} = \{j \in \mathcal{N}|j \in f_\gamma, j \neq \underline{\ddot{f}_\gamma}\}$.

The $\mathcal{SPPFP}$ consists of finding the cheapest way to connect o to each other node $j \in \mathcal{N} - \{o\}$, such that no paths in the solution contain any forbidden path $f_\gamma, \gamma = 1, \ldots, |\mathcal{F}|$.

The $\mathcal{SPPFP}$ can be viewed as a particular instance of the $\mathcal{SPPRC}$. More specifically, it is assumed that $|\mathcal{F}|$ resources are available, one for each forbidden path, and we denote the resource limit vector by $R = \{R_1 = n_{f_1} - 2, \ldots, R_{|\mathcal{F}|} = n_{f_{|\mathcal{F}|}} - 2\}$, where n_{f_γ} represents the number of nodes of the forbidden path $f_\gamma, \gamma = 1, \ldots, |\mathcal{F}|$.

In addition, let $w_{ij} = \{w_{ij}^1, \ldots, w_{ij}^{|\mathcal{F}|}\}$ represent the resource consumed along the arc $(i, j) \in \mathcal{A}$, where $\forall f_\gamma \in \mathcal{F}, w_{ij}^\gamma = 1$ if $(i, j) \in f_\gamma$, 0 otherwise.

Given a path $\Pi_{od} = \{o = i_1, \ldots, i_l = d\}$ from the origin o to the destination d, the resources consumed along Π_{od} are stored in the vector $W_d = \{w^1(\Pi_{od}), \ldots, w^{|\mathcal{F}|}(\Pi_{od})\}$, where for each forbidden path $f_\gamma, \gamma = 1, \ldots, |\mathcal{F}|, w^\gamma(\Pi_{od})$ represents the number of consecutive arcs of f_γ, starting from $\dot{f}_\gamma$, that are included in Π_{od}.

The $\mathcal{SPPFP}$ is the problem of finding a path Π_{oj} in $\mathcal{D}$ from o to each other node $j \in \mathcal{N} - \{o\}$, for which the costs $z(\Pi_{oj})$ are minimized, subject to the constraints that $w^\gamma(\Pi_{oj}) \leq R_\gamma$ for all $\gamma = 1, \ldots, |\mathcal{F}|$ and $\forall j \in \mathcal{N} - \{o\}$.

A feasible solution for the $\mathcal{SPPFP}$ can contain repeated nodes, even if the graph does not present negative cost cycles. This situation can be explained by taking into account the specific structure of the constraints of the $\mathcal{SPPFP}$. Indeed, the generation of a cycle allows avoiding that the built path solution contains a specific forbidden path. This concept is better underlined in the example of Figure 3.1.

3.3 Solution approach

Dynamic programming approaches, based on the algorithm proposed by Desrochers ([50]), are developed to find the optimal solution for the $\mathcal{SPPFP}$. The main differences with the algorithm of Desrochers are related to the rule for constructing a new label/state and the strategy used to exploit the solution space. In particular, the resources

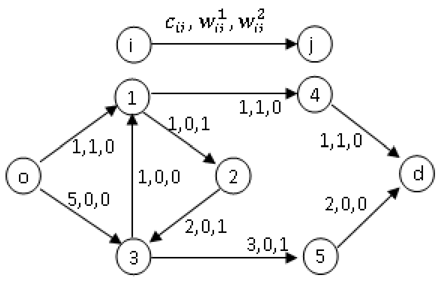

Figure 3.1: The set of forbidden paths contains the paths $f_1 =< o, 1, 4, d >$ and $f_2 =< 1, 2, 3, 5 >$. With each arc (i, j) the cost c_{ij}, the resource consumption w_{ij}^1 and w_{ij}^2 are associated and related to f_1 and f_2, respectively. The optimal path from node o to node d is $\Pi_{od}^* = \{o, 1, 2, 3, 1, 4, d\}$, with cost 7.

consumption of a partial solution, is not defined as the sum of the resources consumed along the partial path but it depends on the number of consecutive arcs, belonging to forbidden paths, that are included in the path. Thus the updated rule takes into account the specific structure of the forbidden paths. In addition, label setting approaches based on both node and label selection strategies are developed. In order to describe the devised solution methods, it is useful to introduce the following notations and definitions.

Let $s_k = (z_k, W_k)$ denote a state, associated with a path Π_{ok} from the origin node o to the node k. In particular, z_k represents the cost of the path Π_{ok} (i.e., $z_k = z(\Pi_{ok})$), whereas W_k is the vector containing information about the resources consumed along Π_{ok}, that is $W_k = \{w^1(\Pi_{ok}), \ldots, w^{|\mathcal{F}|}(\Pi_{ok})\}$. It is worth observing that more than one path Π_{ok} can exist to reach the node k. In what follows, the set of states associated with each path from node o to node k is denoted as D_k.

Since a partial path is associated with each state, in what follows the terms "state", "path" and "solution" are used in an interchangeable fashion.

Definition 3.3.1. *Let s_k^1 and s_k^2 be two labels associated with the node*

k, we say that z_k^1 dominates z_k^2 if the following conditions hold

$$z_k^1 \leq z_k^2; \tag{3.1}$$

$$W_k^1 \leq W_k^2; \tag{3.2}$$

and at least one of the inequalities is strict.

Definition 3.3.2. *A state is said to be efficient/non-dominated if there is not another state that dominates it.*

Definition 3.3.3. *A state $s_k = (z_k, W_k)$ is said to be feasible if $W_k \leq R$.*

Starting from the state $s_o = (0, 0)$, the solution space is explored in order to obtain efficient solutions for each node. During the course of the algorithm, a state s_h is selected and a new state $s_k = (z_k, W_k), k \in FS(h)$, is determined on the basis of the conditions reported below:

$$z_k = z_h + c_{hk} \tag{3.3}$$

$$w^\gamma(\Pi_{ok}) = \begin{cases} w^\gamma(\Pi_{oh}) + 1 & if \;\; w^\gamma(\Pi_{oh}) \geq 1 \wedge w_{hk}^\gamma = 1 \wedge Cons(\ddot{\Pi}_{oh}, (h,k), f_\gamma) \\ w^\gamma(\Pi_{oh}) + 1 & if \;\; w^\gamma(\Pi_{oh}) = 0 \wedge w_{hk}^\gamma = 1 \wedge \dot{f}_\gamma \equiv (h,k) \\ 0 & if \;\; w^\gamma(\Pi_{oh}) \geq 1 \wedge w_{hk}^\gamma = 1 \wedge !Cons(\ddot{\Pi}_{oh}, (h,k), f_\gamma) \\ 0 & if \;\; w^\gamma(\Pi_{oh}) \geq 0 \wedge w_{hk}^\gamma = 0 \end{cases},$$

$$\gamma = 1, \ldots, |\mathcal{F}| \tag{3.4}$$

where the function $Cons(\ddot{\Pi}_{oh}, (h,k), f_\gamma)$ returns true if the arc (h,k) follows immediately the last arc of the path Π_{oh} (i.e.: $\ddot{\Pi}_{oh}$) in the forbidden path f_γ, false otherwise. The newly created state is then added to the set D_k only if it is feasible, that is $W_k \leq R$, and efficient. To better explain the condition (3.4), let us consider the network of Figure 3.2. We assume that $\mathcal{F} = \{f\}$, with $f =< 3, 1, 4 >$, and we consider the two paths $\Pi_1 = \{o, 1, 2, 3, 1, 4, 5, 1, d\}$ and $\Pi_2 = \{o, 1, 4, 5, 1, 2, 3, 1, d\}$. It is easy to verify that path Π_1 is infeasible, while path Π_2 is feasible.

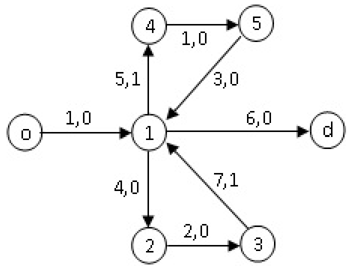

Figure 3.2: Simple Graph Example. Associated with each arc $(i, j) \in \mathcal{A}$ are the cost c_{ij} and the resource consumption w^1_{ij}. The resource limit is $R_1 = 1$.

When the partial solution $\check{\Pi}_1 = \{o, 1, 2, 3\}$ is extended to the arc $(3, 1)$, since $w^1(\check{\Pi}_1) = 0$, $w^1_{31} = 1$ and $(3, 1) = \dot{f}$, the resource consumption is set equal to one, that is $w^1(\check{\Pi}_1 \cup (3, 1)) = 1$. The extension of the new built sub-path $\check{\Pi}_1 \cup (3, 1)$ to the arc $(1, 4)$, generates the state $s_4 = (z_4, W_4) = (19, 2)$, because the first condition of (3.4) holds, that is $w^1(\check{\Pi}_1 \cup (3, 1)) \geq 1$, $w^1_{14} = 1$ and $Cons((3, 1); (1, 4); f) = true$. Since the state s_4 is associated with an infeasible path (i.e.: $W_4 = 2 > 1 = R_1$), thus it is not added to the set D_4.

Let us consider the path Π_2. When we extend the partial path $\check{\Pi}_2 = \{o, 1\}$ to the arc $(1, 4)$, the resource consumption is not updated even if arc $(1, 4)$ belongs to f. This is ensured by the condition $f \not\equiv (1, 4)$. The resource consumption remains equal to zero until the sub-path $\check{\Pi}'_2 = \{o, 1, 4, 5, 1, 2, 3\}$ is extended to the arc $(3, 1)$. Since $f \equiv (3, 1)$, $w^1(\check{\Pi}'_2 \cup (3, 1)) = 1$, when the sub-path $\check{\Pi}'_2 \cup (3, 1)$ is extended to the arc $(1, d)$, the consumption of resource is set equal to zero.

For the sake of comprehension and for a better explanation of the main ideas of the basic solution approach, in Figure 3.3 we illustrate the states associated with each node, when considering the instance depicted in Figure 3.1.

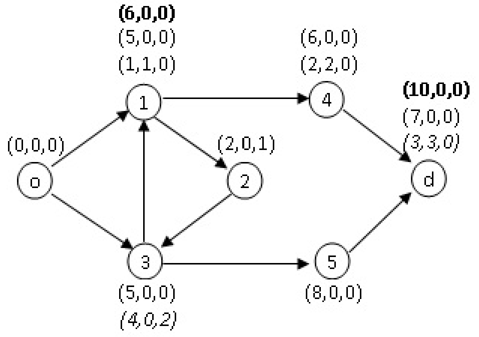

Figure 3.3: Associated with each node $i \in \mathcal{N}$, are the states related to the instance of Figure 3.1. In particular, the dominated states are reported in bold, while the italics represent the infeasible states. All the other states, that is the feasible and efficient one, are stored in the set D_i.

It is worth observing that during the search process, it may occur that some paths have a resource consumption equal to zero. In this case, when we compare a new state with an existing one, on the basis of the conditions given in Definition 1, the condition on the resources is satisfied as equality. Thus the dominance is related only to the cost, that is the state with the minimum cost is the only efficient state. The aforementioned considerations are formally stated in the following lemma.

Lemma 3.3.4. *Let I be the set that contains all nodes that do not belong to any forbidden path, that is $I = \{i \in \mathcal{N} | i \notin f, \forall f \in \mathcal{F}\}$, and let J denote the set containing the destination nodes of all forbidden paths that do not belong to $N^{(\gamma)}, \gamma = 1, \ldots, |\mathcal{F}|$, that is $J = \{j \in \mathcal{N} | \exists f_{\overline{\gamma}} \in \mathcal{F}, \ddot{f}_{\overline{\gamma}} = j, j \notin N^{(\gamma)}, \gamma = 1, \ldots, |\mathcal{F}|\}$. We show that each node $k \in I \cup J$, is associated with exactly one state.*

Proof. For each node $k \in J$, a state s_k exists only if it is feasible, that is $W_k \leq R$. Since $k \in J$, from conditions of (3.4), we have that $W_k = \{0, \ldots, 0\}$, thus all the states associated with nodes k have a resource consumption equal to zero. The efficient one is that with a minimum cost.

Let us now consider the set I. For each node $k \in I$, $w_{hk}^{\gamma} = 0, \gamma = 1, \ldots, |\mathcal{F}|, (h,k) \in \mathcal{A}$. Thus, from the fourth condition of (3.4), each state s_k associated with the node k, is characterized by $W_k = \{0, \ldots, 0\}$. Also, in this case the efficient state is that with minimum cost. $\square$

It is possible to show that after applying the dominance test of Definition 3.3.1, the set $D_k, \forall k \in \mathcal{N}$ is an efficient/non-dominated set, that is, it contains only non-dominated and feasible states.

For each node $j \in \mathcal{N}$, the value $z_j = \min_{s_j \in D_j} z_j$ represents the cost of the shortest path, without forbidden paths, from node o to node j.

With the aim of reducing the number of generated states, a bounding procedure has been devised. In particular, upper bound (UB) and lower bound (LB) on the cost are introduced to prune those states that do not have the potential to generate optimal solutions. In particular, a LB_{ij} is associated with each pair of nodes $i, j \in \mathcal{N}$ and it represents the least cost of the path from node i to node j. The UB_j is associated with each node $j \in \mathcal{N}$ and it denotes the cost of the best feasible solution from node o to node j, found so far.

A state $s_k = (z_k, W_k)$ is generated only if $\exists j \in \mathcal{N} | z_k + LB_{kj} \leq UB_j$. In addition, the UB_j are updated whenever a better feasible solution is generated during the search process.

In order to initialize the UB_j, feasible initial solutions are obtained by applying the heuristic strategy reported in [75]. Briefly, the procedure is divided into two main steps. In the former step, the original graph is modified by deleting one arc for each forbidden path. In the latter, the shortest paths are calculated in the modified graph.

The solution approaches developed to address the $\mathcal{SPPFP}$ have been defined by considering different state selection and node selection strategies. In the first case, at each iteration, a state is selected and a new state is determined, extending the one selected, for each of its successor nodes. In the node-selection method, at each iteration, a node h is selected and all the states associated with it that have not yet extended, are used to update the set of states of the successor nodes.

3.3.1 State selection method

The state selection method ($\mathcal{SSM}$, for short) maintains a candidate list L, storing all the states with the potential to determine a new state for at least one node. At each iteration, a state s_h^ξ is selected and processed. Starting from the selected state, efficient and feasible states are generated and stored. The steps of the proposed algorithm are depicted in Algorithm **3**.

It is possible to show that $\mathcal{SSM}$ determines the set of all efficient and feasible solutions associated with each node.

Proposition 3.3.5.

1. *At the end of each iteration, the following conditions hold:*

   (a) $D_o = \{s_o^1\} = \{(0,0)\}$*;*

   (b) $\forall j \in \mathcal{N}$*, if* $D_j \neq \emptyset$ *[i.e.:* $D_j = \{s_j^1, s_j^2, \ldots, s_j^l\}$*] and* $j \neq o$*, then* $s_j^\xi, \xi = 1, \ldots, l$ *is a state related to a feasible path from node* o *to node* j *and* D_j *is an efficient set.*

2. *Upon termination of the algorithm, if* $D_j \neq \emptyset, j \in \mathcal{N}$ *and* $j \neq o$*, then* D_j *contains the states of all efficient and feasible paths from node* o *to node* j*.*

Proof.

Algorithm 3 $\mathcal{SSM}$ scheme

```
Step 0 (Initialization Phase)
Set: $D_o = \{s_o^1\}, s_o^1 = (0,0), D_k = \emptyset, \forall k \in \mathcal{N} - \{o\}, L = \{s_o^1\}$.

Step 1 (State Selection)
if $L = \emptyset$ then
  STOP
else
  Select and delete from $L$ a state $s_h^\xi$.
end if

Step 2 (State Scan)
for all $k \in FS(h), k \neq 0$ do
  Compute $\overline{W}_k$ on the basis of the condition (3.4).
  if $\overline{W}_k \leq W$ and $\exists j \in \mathcal{N} | z_h^\xi + c_{hk} + LB_{kj} \leq UB_j$ then
    Set: $\overline{z}_k = z_h^\xi + c_{hk}$.
    if $\overline{s}_k = (\overline{z}_k, \overline{W}_k)$ is not dominated by the states belonging to $D_k$ then
      Set: $s_k^{|D_k|+1} = (\overline{z}_k, \overline{W}_k)$; $D_k = D_k \cup \{s_k^{|D_k|+1}\}$.
          $L = L \cup s_k^{|D_k|+1}$.
          Delete from $D_k$ and $L$ all the states that are dominated by $s_k^{|D_k|+1}$.
      if $UB_k > \overline{z}_k$ then
        Set: $UB_k = \overline{z}_k$.
      end if
    end if
  end if
end for
Go to Step 1.
```

1. Condition 1a holds because, initially, $D_o = \{s_o^1\} = \{(0,0)\}$ and, by the rules of the algorithm, D_o cannot change.

   We prove condition 1b by induction on the iteration count. Indeed, initially, condition 1b holds, since node o is the only node for which set D_o is nonempty. Suppose that 1b holds for some node j at the beginning of some iteration.

   Let s_i^ξ be the state removed from L.

   If $i = o$, it occurs only at the first iteration and $\xi = 1$. At the end of this iteration, we have $D_j = \{s_j^1\}$ for all successor nodes j of o such that the corresponding path Π_{oj} is feasible and s_j^1 has the potential to generate an optimal path for at least one node (i.e.: $\exists k \in \mathcal{N} | z_j^1 + LB_{jk} \leq UB_k$). $D_j = \emptyset$ for all other nodes $j \neq o$, $j \notin FS(i)$. Thus, the set of states has the required property.

If $i \neq o$, then s_i^ξ is the state associated with some feasible and potential optimal path Π_{oi}^ξ starting from o and ending with i which is not dominated by the other paths in D_i, by the induction hypothesis. If the set D_j changes, for some node j, such that $j \in FS(i)$, as a result of the iteration, a new state $\bar{s}_j = (\bar{z}_j, \bar{W}_j)$ such that $\bar{W}_j \leq R$ and $\exists k \in \mathcal{N} | \bar{z}_j^1 + LB_{jk} \leq UB_k$ is obtained for node j. The created state is related to the feasible path Π_{oj} consisting of path Π_{oi}^ξ followed by the arc (i, j). Finally, note that, by the rules of the algorithm, the newly created label is added to D_j only if it is an efficient label. This completes the induction proof of 1b.

2. Using part 1b, we have that, at each iteration, $\forall j \in \mathcal{N}$ such that $D_j \neq \emptyset$, D_j is an efficient set. Thus, the property mentioned is also satisfied when the algorithm terminates. In addition, the way in which the candidate list L is updated and the termination condition (i.e., the algorithm terminates when there are no more states left to be scanned) guarantees that all the states with the potential to determine a new state for at least one node are scanned during the execution of the algorithm.

$\square$

3.3.2 Node selection method

The node selection approach ($\mathcal{NSM}$, for short) maintains a candidate list L, containing the nodes to be processed. At each iteration a node h is selected and efficient and feasible states are generated from the states belonging to D_h that have not yet extended to the successor nodes of h, in the previous iterations. In Algorithm **4**, we present the steps of the proposed $\mathcal{NSM}$.

Algorithm 4 $\mathcal{NSM}$ scheme

```
Step 0 (Initialization Phase)
Set: $D_o = \{s_o^1\}, s_o^1 = (0,0), D_k = \emptyset, \forall k \in \mathcal{N} - \{o\}, L = \{o\}$.

Step 1 (Node Selection)
if $L = \emptyset$ then
  STOP
else
  Select and delete from $L$ a node $h$.
end if

Step 2 (Node Process)
for all $k \in FS(h), k \neq o$ do
  for all $s_h^\xi \in D_h$ not yet extended do
    Compute $\overline{W}_k$ on the basis of the condition (3.4).
    if $\overline{W}_k \leq W$ and $\exists j \in \mathcal{N} | z_h^\xi + c_{hk} + LB_{kj} \leq UB_j$ then
      Set: $\overline{z}_k = z_h^\xi + c_{hk}$.
      if $\overline{s}_k = (\overline{z}_k, \overline{W}_k)$ is not dominated by the states belonging to $D_k$ then
        Set: $s_k^{|D_k|+1} = (\overline{z}_k, \overline{W}_k)$; $D_k = D_k \cup \{s_k^{|D_k|+1}\}$.
            $L = L \cup s_k^{|D_k|+1}$.
            Delete from $D_k$ and $L$ all the states that are dominated by $s_k^{|D_k|+1}$.
        if $UB_k > \overline{z}_k$ then
          Set: $UB_k = \overline{z}_k$.
        end if
      end if
    end if
  end for
end for
Go to Step 1.
```

On the basis of the operations executed by $\mathcal{NSM}$, the following proposition can be demonstrated.

Proposition 3.3.6.

1. *At the end of each iteration, the following conditions hold:*

   (a) $D_o = \{s_o^1\} = \{(0,0)\}$*;*

   (b) $\forall j \in \mathcal{N}$*, if* $D_j \neq \emptyset$ *[i.e.:* $D_j = \{s_j^1, s_j^2, \ldots, s_j^l\}$*] and* $j \neq o$*, then* $s_j^\xi, \xi = 1, \ldots, l$ *is a state related to a feasible path from*

node o to node j and D_j is an efficient set.

2. *Upon termination of the algorithm, if $D_j \neq \emptyset, j \in \mathcal{N}$ and $j \neq o$, then D_j contains the states of all efficient and feasible paths from node o to node j.*

Proof.

1. The proof of condition 1a is identical to the corresponding part of Proposition 1.

   We prove condition 1b by induction on the iteration count. Initially, condition 1b holds, since node o is the only node for which the set D_o is nonempty. Suppose that 1b holds for some node j at the start of some iteration at which the node removed from L is i. The following alternative situations can occur:

   - $i = o$. This situation happens only at the first iteration and at the end of the iteration we have

   $$D_j = \begin{cases} s_j^1 = (z_j^1, w_j^1), & \text{if } w_j^1 \leq \mathcal{R}, \exists k \in \mathcal{N} | z_j^1 + LB_{jk} \leq UB_k \\ & \text{and } j \in FS(i) \\ \emptyset, & \text{otherwise} \end{cases}$$

     Thus, the set of labels D_j has the required property.

   - $i \neq o$. In this case, $D_i \neq \emptyset$ by the rules of the algorithm, and it contains the states of some paths Π_{oi}^l, $l = 1, \ldots, |D_i|$, starting from node o and ending at node i. In addition, we have that D_i is an efficient set, by the induction hypothesis. If by using the states belonging to D_i, a new set $\bar{D}_j$ is obtained, for some node j, such that $j \in FS(i)$, as a result of the iteration, $\bar{D}_j$ is set to $D_j \cup \{\bar{s}_j^1, \ldots, \bar{s}_j^m\}$, $m \leq |D_i|$, where $\bar{s}_j^l, l = 1, \ldots, m$ is not dominated by the labels already belonging to D_j and the corresponding path Π_{oj}^l, consisting

of Π^l_{oi} followed by the arc (i, j), is feasible and has the potential to determine optimal solutions. In addition, by the rules of the algorithm, the states dominated by the newly created states $\bar{s}^l_j, l = 1, \ldots, m$ are deleted from D_j.
Thus, property 1b holds at the end of the iteration, completing the induction proof.

2. The proof is identical to the corresponding part of Proposition 3.3.5.

□

It is important to point out that the algorithm proposed by Desrochers in [50] can be easily derived from the $\mathcal{NSM}$ scheme (see Algorithm **4**). In particular, it is sufficient to adopt the Bellman-Ford strategy to select, at each iteration, the node h to be processed (**Step 1**), to extend all the labels associated with node h and to eliminate the pruning rule (**Step 2**) when a new state is determined.

As far as the complexity analysis of the proposed solution approaches is concerned, the following theoretical results can be derived.

Proposition 3.3.7. *Given the directed graph $\mathcal{D}(\mathcal{N}, \mathcal{A})$ and let k denote the number of efficient and feasible paths, that reach each node $j \in \mathcal{N}$ starting from node o, the computational cost of the proposed $\mathcal{SSM}$ and $\mathcal{NSM}$ when applied to $\mathcal{D}(\mathcal{N}, \mathcal{A})$ is $\mathcal{O}(k^2|\mathcal{A}|\,|\mathcal{F}|)$, in the worst case.*

Proof. Both methods execute the same number of operations. In particular, it is easy to verify that the number of states/nodes to be explored can never be greater than $k\,|\mathcal{N}|$. Thus, the for-loop (**Step 2**) is performed $k\,|\mathcal{N}|$ times and requires the execution of $\frac{|\mathcal{A}|}{|\mathcal{N}|}$ iterations. At each iteration, the calculation of the amount of resource consumed takes $\mathcal{O}(4\,|\mathcal{F}|)$, while verifying label dominance takes $\mathcal{O}(k\,|\mathcal{F}|)$, in the worst case. The feasibility check requires $\mathcal{O}(|\mathcal{F}|)$ whereas $\mathcal{O}(|\mathcal{N}|)$ operations are required by the pruning rule. Finally, adding/replacing a label in the set $D(.)$ takes $\mathcal{O}(1)$, whereas the computational cost for adding an element into L is $\mathcal{O}(1)$. Thus, the computational cost of the for-loop is equal to $\mathcal{O}(\frac{|\mathcal{A}|}{|\mathcal{N}|} \times max(5\,|\mathcal{F}|\,; k\,|\mathcal{F}|\,; |\mathcal{N}|))= \mathcal{O}(\frac{|\mathcal{A}|}{|\mathcal{N}|}k\,|\mathcal{F}|)$. On the basis of the previous considerations, it is evident that the worst case computational complexity of the proposed approaches is $\mathcal{O}(k^2|\mathcal{A}|\,|\mathcal{F}|)$. $\square$

3.4 Computational Experiments and Discussion

In this section we report, discuss and compare the results of the computational experiments carried out with our implementation of $\mathcal{SSM}$ and $\mathcal{NSM}$.

The computational experiments were designed to study the performance of these two methods relative to each other as well as the dependence of performance on network topology and number of forbidden paths. The implemented algorithms have been also compared with (1) the algorithm of Desrochers ([50]), that represents the starting point for the development of the methods presented here; (2) an enhanced version of the same approach, that uses the pruning rule; (3) the algorithm of Villeneuve and Desaulniers ([138]) ($\mathcal{VD}$, for short) and the solution approach proposed in [30] ($\mathcal{EVD}$, for short) that represent the state-of-art approaches to address the problem under investigation.

The solution approaches have been implemented in java language and have been tested by using an Intel(R) Core(TM) i7 CPU M 620 PC, 2.67 GHz, RAM 4.00 GB, under Microsoft 7 operating system.

It is worth observing that the codes implementing $\mathcal{VD}$ and $\mathcal{EVD}$ have been provided by the authors of [30]).

3.4.1 Test Problems

The computational experiments were conducted on three sets of test problems. The first set (i.e., sprand networks) contains 16 test problems ($R_1 - R_{16}$) with different size and density, defined as the number of arcs over the number of nodes. These instances have been generated by using the public domain SPRAND generator ([29]). In addition, we have considered a set of 4 complete networks (i.e., $C_1 - C_4$) built by using the Compligen generator of Bertsekas ([20]).

The characteristics of the sprand networks are reported in Table 3.1 where the number of nodes, the number of arcs and the density values are highlighted, whereas in Table 3.2, the number of nodes and the number of arcs of the complete networks are given. The costs have been randomly generated from the interval $[0, 10]$ in the case of sprand networks and from the range $[0, 100]$ for the complete networks.

Test	Nodes	Arcs	Density
R_1	100	5000	50
R_2	200	5000	25
R_3	500	5000	10
R_4	1000	5000	5
R_5	200	10000	50
R_6	400	10000	25
R_7	1000	10000	10
R_8	2000	10000	5
R_9	400	20000	50
R_{10}	800	20000	25
R_{11}	2000	20000	10
R_{12}	4000	20000	5
R_{13}	600	30000	50
R_{14}	1200	30000	25
R_{15}	3000	30000	10
R_{16}	6000	30000	5

Table 3.1: Characteristics of the sprand networks.

For each network, several instances have been created by setting the total number of forbidden paths (i.e., $|\mathcal{F}|$) equal to 20, 60, 100, 250 and 500 for networks R_1-R_{16} and to 20, 40, 60, 80 and 100 in the case

Test	Nodes	Arcs
C_1	150	22350
C_2	180	32220
C_3	200	39800
C_4	250	62250

Table 3.2: Characteristics of the complete networks.

of tests C_1-C_4, respectively. For each instance, we have considered six different problems, characterized by a different length (i.e., number of nodes) of the forbidden paths. In particular, forbidden paths with 5, 6, 7, 8, 9 and 10 nodes have been generated.

In what follows, we use the notation $test^{|\mathcal{F}|}$ to represent a test problem. Indeed, R_1^{250} refers to the fully random network R_1, with 250 forbidden paths.

We underline that the forbidden paths have been generated in such a way that all of them are included in the optimal solution of the problem without the forbidden paths restriction. All these paths produce violated forbidden constraints in the $\mathcal{SPPFP}$ relaxation and, as a consequence, a large number of states has many positive resources consumption (for counting arcs of forbidden paths). In this manner we are sure to evaluate the proposed algorithms on meaningful instances.

In addition, it is worth observing that if a small fraction of forbidden path constraints is binding, a simple algorithm based either on $\mathcal{VD}$ or on $\mathcal{EVD}$ could outperform the proposed methods. In particular, the shortest paths on the original graph are determined. For all forbidden paths that make the determined optimal solution infeasible, the enlarged network using either $\mathcal{VD}$ or $\mathcal{EVD}$ is constructed and the shortest paths are re-computed. This procedure is repeated until for each node the shortest path is a feasible path for the $\mathcal{SPPFP}$, thus the optimal solution is obtained. Of course, the approach outlined above turns out to be very inefficient for the test problems considered here.

In order to compare the proposed solution approaches with the state-of-art algorithms, another set of test problems (i.e. $N_1 - N_9$) has been considered. In particular, the instances tested in [30] have been taken into account in the computational test. The characteristics (i.e., nodes, arcs and density) of these test problems (referred in the sequel as fully random networks) are reported in Table 3.3.

Test	Nodes	Arcs	Density
N_1	100	1000	10
N_2	100	5000	50
N_3	100	9000	90
N_4	300	9000	30
N_5	300	45000	150
N_6	300	81000	270
N_7	500	25000	50
N_8	500	125000	250
N_9	500	225000	450

Table 3.3: Characteristics of the fully random networks.

For each network $N_i, i = 1, \ldots, 9$, six different instances have been created, by letting the number of forbidden paths equal to 250, 500 and 1000 and by considering forbidden paths with 10 and 20 nodes .

3.4.2 Test Codes

In our testing, we compared the following twelve codes:

$\mathcal{SSM} - ES_s - nP$ This code implements the $\mathcal{SSM}$ scheme, in which the state, to be processed at each iteration, is selected by following a Dijkstra-Like rule (ES_s, for short). More specifically, at each iteration of $\mathcal{SSM}$, the chosen state s_h^ξ is the one with the smallest cost, that is $s_h^\xi = arg\min_{s_h \in L} z_h$.

$\mathcal{SSM} - ES_s - P$ This code differs from the previous one only in the use of the pruning strategy. In this version, the information related to the UBs and LBs are taken into account to eliminate unpromising states.

$\mathcal{SSM} - FIFO - nP$ Same as $\mathcal{SSM} - ES_s - nP$, except that the state to be processed is selected following the Bellman-Ford rule ($FIFO$, for short).

$\mathcal{SSM} - FIFO - P$ Same as $\mathcal{SSM} - FIFO - nP$, except that the pruning strategy is used to fathom unpromising states.

$\mathcal{NSM} - ES_n - nP$ This code implements the $\mathcal{NSM}$ scheme. A node is selected by adopting the Dijkstra-Like rule (ES_n, for short), described in what follows. For each node the smallest cost $c(h)$ among all the states belonging to D_h is determined, that is $c(h) = \min_{s_h \in D_h}\{z_h\}$. The node $\bar{h}$ satisfying the condition $\bar{h} = arg \min_{h \in L}\{c(h)\}$ is selected.

$\mathcal{NSM} - ES_n - P$ The ES_n selection strategy is followed to select, at each iteration of $\mathcal{NSM}$, the node to be processed and the pruning rule is applied to fathom unpromising nodes.

$\mathcal{NSM} - FIFO - nP$ This code is a $\mathcal{NSM}$, based on the $FIFO$ node selection strategy. Since the pruning rule is not applied, all the possible feasible states are generated and explored.

$\mathcal{NSM} - FIFO - P$ An enhanced version of $\mathcal{NSM} - FIFO - nP$, that considers the pruning rule.

$\mathcal{D}$ This code implements the Desrochers' algorithm.

$\mathcal{D} - P$ An enhanced version of $\mathcal{D}$, that considers the pruning rule.

$\mathcal{VD}$ This code implements the procedure defined by Villeneuve and Desaulniers ([138]).

$\mathcal{EVD}$ This code implements the algorithm proposed in [30].

A detailed accounting of the collected computational results is given in Tables A.1 - A.10 of Appendix A, where the number of iterations and the execution times (in ms) are given.

It is important to observe that in the case of $\mathcal{SSM}_s$, an iteration corresponds to the execution of the scanning of a state, whereas for the $\mathcal{NSM}_s$ an iteration corresponds to the processing of a node.

3.4.3 Computational Results on sprand Networks

A first phase of the experimental tests has been carried out to investigate the influence of the selection strategies on the computational cost. A significant observation that can be made from the results reported in Tables A.1-A.5 of Appendix A is that ES is on average 2 times faster than $FIFO$. In particular, the average execution time is equal to 14691.52 when the Dijkstra-like selection rule is adopted, whereas it increases to 29916.95 when the Bellman-Ford strategy is applied. This behaviour can be explained by taking into account the number of iterations executed by the compared algorithms. Indeed, the approaches based on $FIFO$ execute, on average, a number of iterations 3.43 times greater than those performed by the remaining codes. The improvement is more evident for the $\mathcal{SSM}_s$, in which the use of the ES_s allows us to reduce the computational effort by 56.22%, whereas in the case of $\mathcal{NSM}_s$ an execution time reduction of 44.43% is observed. On the basis of the considerations reported above, in what follows we restrict our attention to the ES versions, that is the ES_s and the ES_n for $\mathcal{SSM}$ and $\mathcal{NSM}$, respectively.

The computational results collected underlined that the performances of the implemented algorithms seem to be strongly affected by the number of forbidden paths. This behaviour is highlighted in Figure 3.4, where we report the average execution time as a function of the number of forbidden paths, obtained with $SSM - ES_s$ and $NSM - ES_n$.

This trend is also underlined by the results reported in Tables 3.4 and 3.5, where the percentage increasing of the execution time in respect of that obtained when solving the instances with 20 forbidden

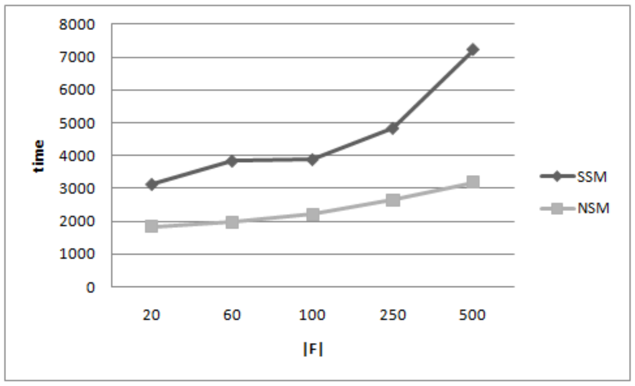

Figure 3.4: Trend of the average execution time with respect to the number of forbidden paths for sprand networks.

paths is shown. For both versions, the worsening of time resolution can be explained by taking into account the number of iterations executed by the methods. In particular, the higher the number of forbidden paths, the higher the number of iterations executed (see Tables 3.4 and 3.5).

$\lvert\mathcal{F}\rvert$	time	% increasing	iter	% increasing
20	3112.30	0.00%	10410.42	0.00%
60	3826.98	22.96%	10705.49	2.83%
100	3869.47	24.33%	11078.99	6.42%
250	4825.87	55.06%	11705.95	12.44%
500	7234.63	132.45%	12378.31	18.90%
AVG	4573.85		11255.83	

Table 3.4: Average execution time, average number of iterations, percentage worsening in time and number of iterations obtained with $SSM - ES_s$ on sprand networks.

$\lvert\mathcal{F}\rvert$	time	% increasing	iter	% increasing
20	1834.72	0.00%	1480.99	0.00%
60	1964.96	7.10%	1485.94	0.33%
100	2201.40	19.99%	1489.63	0.58%
250	2636.01	43.67%	1505.93	1.68%
500	3185.35	73.61%	1542.25	4.14%
AVG	2364.49		1500.95	

Table 3.5: Average execution time, average number of iterations, percentage worsening in time and number of iterations obtained with $NSM - ES_n$ on sprand networks.

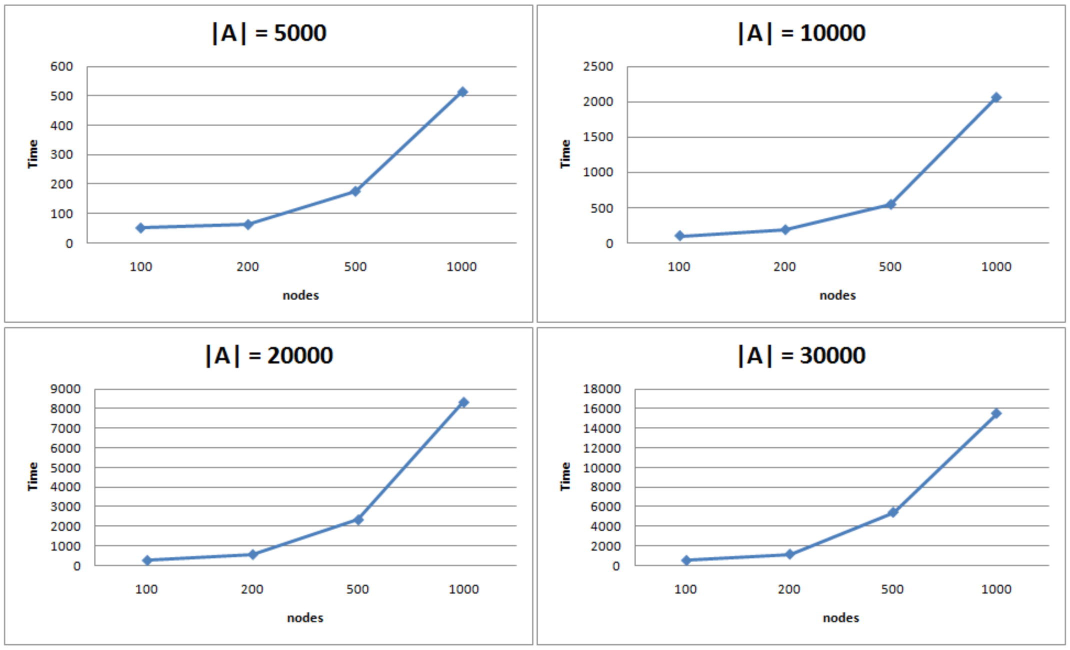

Figure 3.5: Trend of the average execution time as a function of the number of nodes for $NSM - ES_n$.

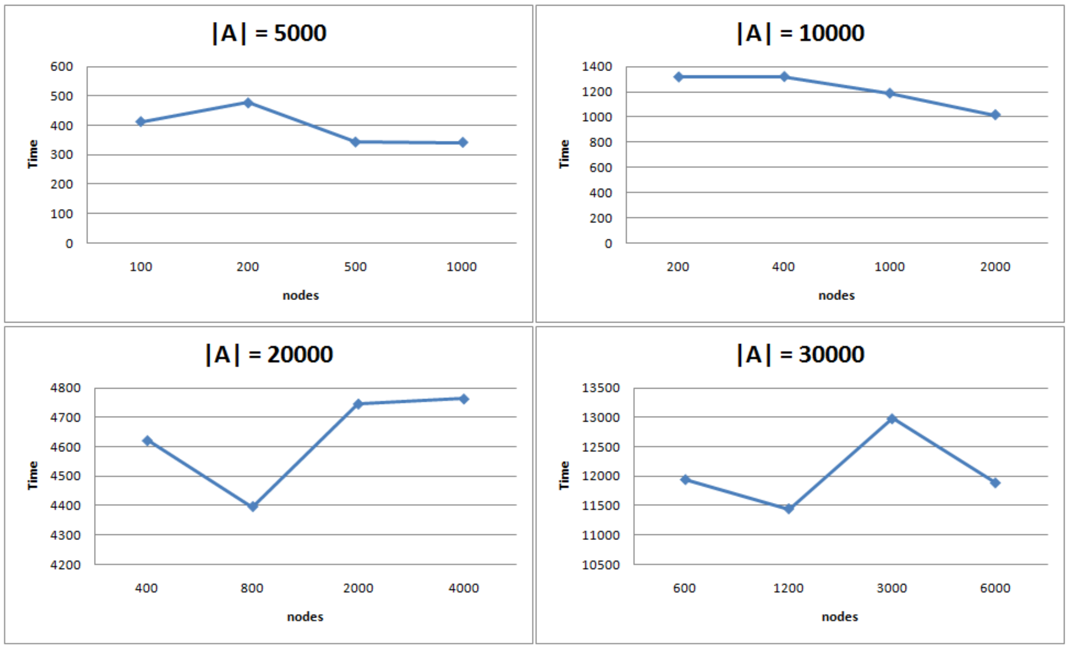

Figure 3.6: Trend of the average execution time as a function of the number of nodes for $SSM - ES_s$.

Figures 3.5 and 3.6 can be used to analyze the influence of the network structure on the computational performance of the proposed solution approaches. In particular, for each set of test problems with the same number of arcs, the average execution time as a function of the number of nodes is plotted.

The computational results collected for $\mathcal{NSM}$, clearly underline a strong relation between the execution time and the structure of the network (see Figure 3.5). In particular the higher the number of nodes, the higher the computational cost. This behaviour can be justified by considering the number of iterations performed by $\mathcal{NSM}$ reported in Table 3.6. Indeed, the higher the number of nodes, the higher the number of iterations.

		$\mathcal{NSM} - ES_n - P$		$\mathcal{NSM} - ES_n - nP$		AVG	
$\#arcs$	$\#nodes$	time	iter	time	iter	time	iter
5000	100	23.92	118.63	81.12	123.87	52.52	121.25
	200	45.76	223.23	82.68	230.8	64.22	227.02
	500	157.56	568.23	196.56	573.83	177.06	571.03
	1000	510.64	1049.63	518.44	1049.63	514.54	1049.63
10000	200	56.16	215.37	140.4	224.3	98.28	219.83
	400	152.36	444.3	214.24	450	183.3	447.15
	1000	502.84	1058	580.84	1064.73	541.84	1061.37
	2000	2037.89	2048.33	2072.21	2048.33	2055.05	2048.33
20000	400	223.08	423.37	326.56	434.3	274.82	428.83
	800	535.08	836.2	619.84	841.53	577.46	838.87
	2000	2157.49	2034.77	2538.66	2040.4	2348.07	2037.58
	4000	8320.05	4043.97	8374.65	4044.2	8347.35	4044.08
30000	600	484.12	622.1	552.76	624.5	518.44	623.3
	1200	1227.73	1235.13	1088.37	1249.9	1158.05	1242.52
	3000	5358.63	3032.9	5487.6	3037.37	5423.11	3035.13
	6000	14601.69	6019.07	16393.63	6019.37	15497.66	6019.22

Table 3.6: Average execution time, average number of iterations, obtained with $NSM - ES_n$ on sprand networks.

A different trend is observed for $\mathcal{SSM}_s$, as shown in Figure. 3.6. Indeed, in this case, the average execution time seems not to be strongly related to the structure of the network. A possible explanation for this behaviour derives from considering the data collected in Table 3.7. In particular, for $\mathcal{SSM} - ES_n - nP$ we have that the higher the density, the higher the computational cost. This specific performance is mainly due to the cost per iteration, that, in this case, is related to the scanning of a state. Table 3.7 highlights that the higher the number of nodes, the higher the number of iterations and the lower the cost per iteration. In particular, at each iteration, a label is selected and extended for all successor nodes. It is worth observing that the lower the density, the lower the cardinality of the forward star. Thus, a lower number of states should be evaluated for each iteration and thus the cost per iteration decreases. The trend is inverted when $\mathcal{SSM} - ES_s - P$ is considered. This behaviour can be justified by taking into account the computational overhead required to execute the pruning rule.

		$\mathcal{SSM} - ES_s - P$		$\mathcal{SSM} - ES_s - nP$		AVG	
$\#arcs$	$\#nodes$	time	iter	time	iter	time	iter
5000	100	68.64	842.67	757.12	4003.53	412.88	2423.10
	200	120.64	1551.07	834.09	5685.87	477.36	3618.47
	500	204.36	2359.13	481.00	5196.03	342.68	3777.58
	1000	299.00	3485.03	383.76	5326.80	341.38	4405.92
10000	200	243.88	1686.03	2387.34	9014.73	1315.61	5350.38
	400	430.04	2743.47	2205.85	9879.50	1317.95	6311.48
	1000	777.92	4631.53	1594.85	10226.97	1186.39	7429.25
	2000	728.52	4702.87	1304.69	10337.10	1016.61	7519.98
20000	400	1075.89	3557.63	8166.13	18944.93	4621.01	11251.28
	800	1735.77	5352.33	7054.89	19930.90	4395.33	12641.62
	2000	3698.78	10156.63	5793.88	20242.93	4746.33	15199.78
	4000	4282.23	12522.30	5244.75	20335.67	4763.49	16428.98
30000	600	2837.14	6132.07	21042.97	29014.87	11940.06	17573.47
	1200	4037.83	8251.80	18845.44	29923.13	11441.63	19087.47
	3000	9414.14	16680.70	16542.35	30230.43	12978.24	23455.57
	6000	8594.10	16904.50	15175.26	30333.43	11884.68	23618.97

Table 3.7: Average execution time, average number of iterations, obtained with $SSM - ES_s$ on sprand networks.

The computational results show that the higher the number of nodes, the higher the computational time. In other words, the proposed solution approaches seem to work better on networks with high density value. Indeed, for $SSM - ES_s$, the average execution time, grouped by the density, is equal to 1568, 20, 2658.91, 6515.12 and 9680.07 for networks with density values equal to 50, 25, 10 and 5, respectively. A similar trend is observed for the $NSM - ES_n$ (i.e., average execution time of 196.82, 490.23, 2044.13 and 5094.06 for increasing density values). This performance can be justified by considering the effect of the fathoming rule. In particular, the lower the number of nodes, the higher the number of fathomed states. To graphically represent this specific behaviour, we report in the following figure (Figure 3.7) the percentage of fathomed states over the total states as a function of the density.

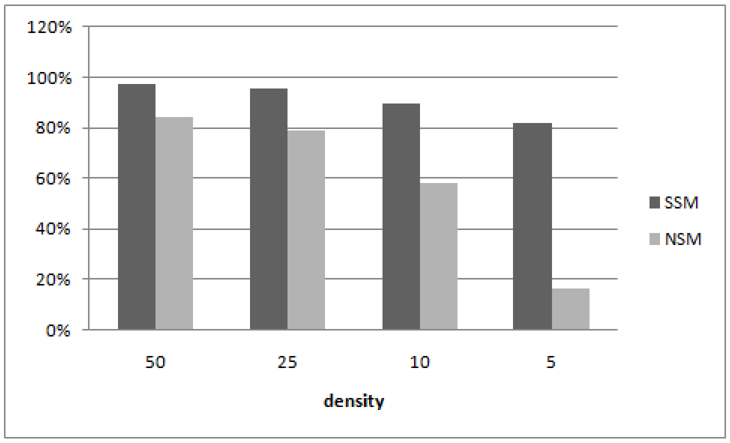

Figure 3.7: The percentage of fathomed states over the total states as a function of the density.

From Figure 3.7, it is evident that the proposed rule is useful to fathom unpromising states. In particular, the average reduction ratio in the number of fathomed states for $SSM - ES_s$ ($NSM - ES_n$) is about 97.39%, 95.46%, 89.38% and 81.58% (84.13%, 78.98%, 57.95% and 16.22%) for networks with density values equal to 50, 25, 10 and 5, respectively.

In addition, it seems that the fathoming rule is more effective for $\mathcal{SSM}$. Indeed, on average $SSM - ES_s - P$ is 2.79 times faster than $SSM - ES_s - nP$, whereas in the case of $\mathcal{NSM}$, the version without the fathoming rule performs on average a number of iterations very close to that executed by $NSM - ES_n - P$ (i.e., 1503.57 and 1498.33, respectively). Consequently, the reduction in the execution times is very limited (2454.29 and 2274.69 ms, for $NSM - ES_n - nP$ and $NSM - ES_n - P$, respectively).

It is important to point out that $\mathcal{NSM} - FIFO - nP$ outperforms its counterpart based on the fathoming strategy. Indeed, $\mathcal{NSM} - FIFO - nP$ is on average 2.32 times faster than $\mathcal{NSM} - FIFO - P$.

This behavior has also been observed for the Desrochers' algorithm (i.e., $\mathcal{D}$), which outperforms its counterpart based on the fathoming strategy. Indeed, $\mathcal{D}$ is on average 1.99 times faster than $\mathcal{D} - P$ (see Table A.6).

The computational results, collected on the sprand networks, clearly underline that $\mathcal{NSM} - ES_n - P$ and $SSM - ES_s - P$ show comparable performance and $\mathcal{NSM} - ES_n - P$ performs the best. In addition, the Desrochers' algorithm (i.e., $\mathcal{D}$) behaves on average slightly worse than $\mathcal{NSM} - ES_n - P$. Indeed, the latter is 1.32 times faster than $\mathcal{D}$.

3.4.4 Computational Results on Complete Networks

In this section, we evaluate the behaviour of the proposed approaches on the set of complete networks. The related computational results are reported in Table A.7 of Appendix A, where, for each test problem, the execution times and the number of iterations required by each algorithm are reported.

As far as the influence of the selection rules on the computational effort is concerned, a behaviour similar to that obtained in the case of sprand networks has been observed. Indeed, ES is on average 4.75 times faster than $FIFO$.

Also in this case, the improvement is more evident for the $\mathcal{SSM}_s$, in which the use of the ES_s determines an average reduction in the computational time of 78.92%, whereas in the case of $\mathcal{NSM}_s$ an execution time reduction of 51.69% is observed. Given the superiority of the Dijkstra-like selection strategy over the $FIFO$ rule, in what follows we concentrate our attention on the computational results obtained with the former (i.e., ES).

The experimental results obtained by testing the proposed approaches on this second set of test problems also underline that, as expected, the execution times increase with the cardinality of the set F. This behaviour is depicted in Figure 3.8, where the average execution time as a function of the number of forbidden paths is reported.

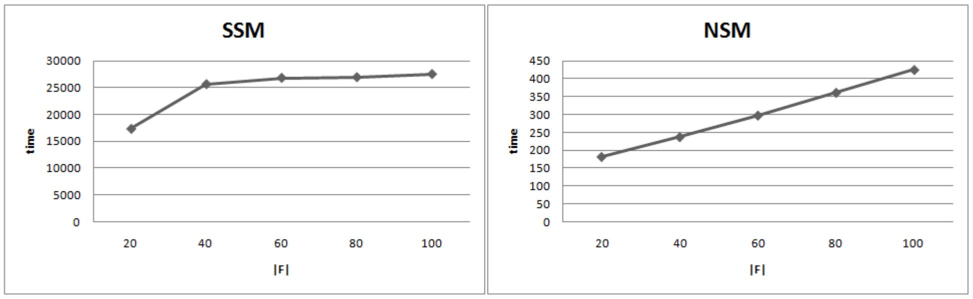

Figure 3.8: Average execution time as a function of the number of forbidden paths on complete networks.

The influence of the cardinality of the set $\mathcal{F}$ on the computational overhead is also underlined by the results reported in Tables 3.8-3.9, where the percentage increasing of the computational effort in respect of that obtained when solving instances with 20 forbidden paths is given. From the results reported in these Tables, it is evident that the worsening of the execution time is mainly due to the increase in the number of iterations.

Another important observation that can be drawn from the results reported in Table A.7 of Appendix A, is that the introduction of the fathoming rule allows an impressive reduction of the computational cost to be obtained. Indeed, the versions based on the pruning strategy are on average 48.89 times faster than the versions that do

$\lvert\mathcal{F}\rvert$	time	% increasing	iter	% increasing
20	17296.29	0.00%	20278.46	0.00%
40	25624.79	48.15%	20288.96	0.05%
60	26799.02	54.94%	20298.06	0.10%
80	26923.82	55.66%	20311.06	0.16%
100	27506.55	59.03%	20325.25	0.23%
AVG	24830.09		20300.36	

Table 3.8: Average execution time, average number of iterations, percentage worsening in time and number of iterations obtained with $SSM - ES_n$ on complete networks.

$\lvert\mathcal{F}\rvert$	time	% increasing	iter	% increasing
20	181.68	0.00%	206.81	0.00%
40	237.25	30.59%	211.48	2.26%
60	297.70	63.86%	215.21	4.06%
80	361.40	98.93%	219.10	5.94%
100	425.43	134.17%	223.08	7.87%
AVG	300.69		215.14	

Table 3.9: Average execution time, average number of iterations, percentage worsening in time and number of iterations obtained with $NSM - ES_n$ on complete networks.

not consider this technique.

Similarly to what was observed for the sprand networks, the fathoming rule is more effective for the $\mathcal{SSM}_s$ than the $\mathcal{NSM}_s$. Indeed, the number of iterations executed by $\mathcal{SSM} - ES_s - P$ is 29.83 times lower than that performed by $\mathcal{SSM} - ES_s - nP$ and the average execution time of the former is 71.66 times lower than the latter. In addition, $\mathcal{NSM} - ES_n - nP$ and $\mathcal{NSM} - ES_n - P$ show similar performance.

The good performance obtained by applying the fathoming rule can be explained by considering the quality of the upper bounds. As shown in Table 3.10, the number of improved upper bound values during the search process is very limited. This result suggests that for a large number of nodes, the heuristic procedure provides the optimal solution.

In what follows, we concentrate our attention on the most efficient

	$\mathcal{SSM}-ES_s$			$\mathcal{NSM}-ES_n$		
	#imp	p.s.	f.s.	#imp	p.s.	f.s.
C_1^{20}	14.50	4629.00	172109.83	5.67	2289.00	61309.17
C_1^{40}	19.83	4657.33	172652.67	6.50	2329.33	62162.83
C_1^{60}	69.33	4811.00	178285.17	13.33	2734.67	69033.67
C_1^{80}	104.50	5050.00	188650.00	24.33	3352.50	78349.17
C_1^{100}	123.33	5235.50	195740.67	33.50	3925.67	86194.50
C_2^{20}	1.50	3406.17	193076.17	1.50	1738.50	75142.00
C_2^{40}	8.50	3540.00	200699.00	0.83	2132.50	85517.83
C_2^{60}	12.17	3564.33	201181.83	2.67	2223.50	88977.00
C_2^{80}	15.33	3579.17	201704.00	3.50	2267.33	90424.83
C_2^{100}	21.17	3660.50	207708.67	5.00	2539.17	97551.67
C_3^{20}	3.17	7862.83	344599.33	1.00	3177.67	108594.00
C_3^{40}	4.67	7906.17	345683.67	2.33	3326.50	113320.67
C_3^{60}	4.67	7945.33	346440.50	2.33	3409.83	115625.33
C_3^{80}	5.83	7978.67	347136.83	3.00	3521.33	118465.67
C_3^{100}	6.00	7994.33	347519.17	3.17	3548.83	119267.33
C_4^{20}	0.50	4803.83	366164.67	0.17	1721.33	102070.17
C_4^{40}	1.50	4826.17	366972.33	0.17	1745.33	103166.67
C_4^{60}	2.17	4845.17	367783.33	0.17	1760.67	104188.83
C_4^{80}	2.17	4861.33	368431.17	0.17	1819.83	107283.67
C_4^{100}	2.83	4881.50	369241.00	0.33	1860.50	109318.00

Table 3.10: Information about the effectiveness of the fathoming rule on sprand networks. #imp represents the times in which the upper bound is updated; p.s. represents the number of states that pass the fathoming test; f.s. indicates the number of states that do not pass the fathoming test.

versions, i.e., those based on the pruning rule.

The computational results obtained by testing $\mathcal{NSM}-ES_n-P$ and $\mathcal{SSM}-ES_s-P$ on complete networks (Table A.7 of the Appendix A) demonstrate clearly that $\mathcal{NSM}-ES_n-P$ is on average 2.09 times faster than $\mathcal{SSM}-ES_s-P$.

The solution approach proposed by Desrochers, that is $\mathcal{D}$ behaves worse than the best performing version of the proposed solution approaches. In particular, $\mathcal{NSM}-ES_n-P$ is 5.51 times faster than Desrochers' algorithm. This behaviour is justified by the number of iterations executed by the two versions. Indeed, $\mathcal{D}$ executes a number of iterations 2.84 times higher than that executed by $\mathcal{NSM}-ES_n-P$.

It is important to point out that the Desrochers' algorithm with the pruning, that is $\mathcal{D}-P$ outperforms the original methods (i.e., $\mathcal{D}$). Indeed, the former is 3.22 times faster than the latter. This behaviour can be related to the number of iterations executed by the methods. Indeed, $\mathcal{D}$ performs, on average, a number of iterations 1.50 times higher than that executed by $\mathcal{D}-P$ (see Table A.8).

3.4.5 Comparison with the state-of-art algorithms

In this section, the best performing version of the proposed approaches (i.e., $\mathcal{NSM}-ES_n-P$) is compared with both the algorithm proposed by Villeneuve and Desaulnirs, that is $\mathcal{VD}$ ([138]) and the improved version, that is $\mathcal{EVD}$ proposed in [30].

It is important to point out that both $\mathcal{VD}$ and $\mathcal{EVD}$ are constructive algorithms. In other words, set $\mathcal{F}$ is used to define a graph, by applying the method proposed in [2]. The obtained graph is then connected to the network by using the technique defined in [100] for solving the $k-\mathcal{SPP}$. In the enlarged graph, the shortest path problem from node o to the other nodes is solved.

The computational results obtained by testing $\mathcal{VD}$ and $\mathcal{EVD}$ on sprand, complete and fully random networks are reported in Tables A.9, A.10 and A.11 of Appendix A, where the execution time needed to construct the enlarged graph and that required to solve the shortest path problem are given.

As far as the comparison of the proposed solution approaches with the state-of-art algorithms, let us separately consider the three sets of test problems.

The computational results obtained by testing $\mathcal{VD}$ and $\mathcal{EVD}$ on the first set of instances (see Table A.9 of Appendix A) underline that the best performing proposed solution approach, that is $\mathcal{NSM}-$

$ES_n - P$, outperforms both $\mathcal{VD}$ and $\mathcal{EVD}$. In particular, $\mathcal{NSM} - ES_n - P$ is on average 2.86 and 2.83 times faster than $\mathcal{VD}$ and $\mathcal{EVD}$, respectively. In adition, it is worth observing that the performance of $\mathcal{EVD}$ and $\mathcal{VD}$ are the same. In particular, the average computational cost is equal to 6448.06 and 6507.07 for $\mathcal{EVD}$ and $\mathcal{VD}$, respectively.

On complete networks (see Table A.10 of Appendix A) , the proposed solution approach outperforms both $\mathcal{VD}$ and $\mathcal{EVD}$. In particular, $\mathcal{VD}$ and $\mathcal{EVD}$ are 35.47 and 35.13 times slower than $\mathcal{NSM} - ES_n - P$. This specific behaviour can be explained by observing that the introduction of the pruning strategy is very effective in fathoming unpromising states when complete networks are considered. Also in this case, $\mathcal{VD}$ and $\mathcal{EVD}$ behave very similar. Indeed, the latter is only 1.01 times faster the the former.

The results collected on the third set of test problems (see Tables A.11 of the Appendix A) clearly underline that the proposed solution approach behaves the best. In particular $\mathcal{NSM} - ES_n - P$ is 62.14 and 78.19 times faster than $\mathcal{EVD}$ and $\mathcal{VD}$, respectively. This behaviour is justified by taken into account the number of iterations executed by the proposed solution approach. As shown in Table A.11, the number of iterastions is very closed to the number of nodes of the networks. This result suggests that the heuristic procedure provides a good upper bound for the considered test problems.

The bad performances of the state-of-art algorithms can be attributed to the computational effort required to build the enlarged graph, while the shortest paths are computed with a minimal computational effort. Indeed, the incidence of the time to compute the shortest path over the total time required by $\mathcal{EVD}$ [and $\mathcal{VD}$] is about 1.93% [and 2.54%], 1.72% [and 2.36%] and 4.14% [and 6.32%] for sprand, complete and fully random networks, respectively.

On the contrary, the proposed solution approaches work directly on the original graph and only requires finding the efficient paths for each node. As a matter of the fact, also when only the time for building the enlarged graph is considered, the proposed solution approach bahaves the best for all the three sets of test problems. Indeed, $\mathcal{NSM} - ES_n - P$ is 2.78 [2.79], 34.53 [34.67] and 59.56 [73.24] times faster than $\mathcal{EVD}$ [$\mathcal{VD}$] for the first, the second and the third set of test problems, respectively.

On the basis of the computational results collected, it is evident that despite the pseudo-polynomial time complexity of the proposed solution approaches, they have very efficient implementation in practice. We remark that both $\mathcal{VD}$ and $\mathcal{EVD}$ has a polynomial complexity. It is worth observing that the good behaviour of the proposed solution approaches is mainly related to the specific characteristic of the $\mathcal{SPPFP}$. Indeed, the result of lemma 3.3.4 suggests that a limited number of efficient paths are associated with each node.

3.5 Conclusions

In this paper, solution methodologies to address the shortest path problem with forbidden paths have been proposed. The developed algorithms are based on the paradigm of the dynamic programming optimization and can be viewed as modified versions of the Desrochers' algorithm ([50]), for the constrained shortest path. A pruning strategy has been defined with the goal to speed-up the search of the optimal solution. With this aim, upper bounds have been determined by developing a well-tailored heuristic procedure. In addition, we have considered both the node and state extension rule and two types of selection strategies. Indeed, a Dijkstra-like and the Bellman-Ford rules have been implemented to select, at each iteration, the node/state to be processed.

The solution approaches presented in this paper have been evaluated numerically on three sets of networks. The first one consists of sprand networks, with different size and density. Also complete networks have been considered and they represent the second set of test problems. The third set contains the fully random networks taken from the literature ([30]). A comparison with the state-of-art algorithm to solve the problem under consideration and with Desrochers' approach has also been made.

On the basis of the obtained results and their analysis, the following conclusions can be drawn:

1) the version that uses the node extension rule with a Dijkstra like selection strategy and the pruning behaves the best;

2) the best performing version is faster than Desrochers' algorithm;

3) the pruning strategy allows a considerable reduction in the computational cost to be obtained. The introduction of the fathoming rule allows us to individuate a large number of unpromising states, especially for networks with high density value;

4) the proposed solution approach is very efficient in solving the shortest path problem with a forbidden path and outperforms the polinomial algorithms which appeared quite recently in scientific literature to address the problem at hand.

Finally, it is worth observing that in this paper we have addressed the non-elementary variant of the shortest path problem with forbidden paths. It could be interesting to investigate the elementary counterpart. This represents the subject of current investigation.

Appendix A - Computational results

		$SS\mathcal{M} - ES_s - P$	$SS\mathcal{M} - ES_s - nP$	$SS\mathcal{M} - FIFO - P$	$SS\mathcal{M} - FIFO - nP$	$\mathcal{N}S\mathcal{M} - ES_n - P$	$\mathcal{N}S\mathcal{M} - ES_n - nP$	$\mathcal{N}S\mathcal{M} - FIFO - P$	$\mathcal{N}S\mathcal{M} - FIFO - nP$
R_1^{20}	Time	59.80	582.40	57.20	1050.41	15.60	23.40	18.20	23.40
	iter	621.00	3980.67	822.67	8265.67	111.00	119.83	186.17	208.17
R_2^{20}	Time	75.40	1183.01	88.40	800.81	23.40	20.80	41.60	33.80
	iter	1077.83	8953.00	1359.50	13052.67	211.17	220.33	424.33	519.33
R_3^{20}	Time	104.00	335.40	208.00	1099.81	83.20	101.40	174.20	106.60
	iter	1450.50	4995.00	3208.67	22089.33	514.50	526.00	1536.83	2069.83
R_4^{20}	Time	208.00	267.80	733.20	1274.01	317.20	304.20	1198.61	278.20
	iter	2829.50	5037.50	11948.50	29413.00	1020.33	1020.33	5195.83	5256.00
R_5^{20}	Time	153.40	1183.01	166.40	2893.82	36.40	33.80	49.40	54.60
	iter	1142.50	8953.00	1315.17	20109.67	209.17	216.33	330.00	459.67
R_6^{20}	Time	197.60	1216.81	234.00	3541.22	80.60	78.00	104.00	119.60
	iter	1402.83	9743.00	1746.00	29432.17	411.17	419.33	699.83	1209.67
R_7^{20}	Time	442.00	1055.61	769.61	4552.63	361.40	314.60	652.60	403.00
	iter	3105.33	9997.67	6153.00	46859.17	1020.50	1025.00	2991.83	4494.33
R_8^{20}	Time	540.80	1008.81	2581.82	5046.63	1424.81	1593.81	3372.22	1159.61
	iter	3827.17	10044.00	21480.33	62812.67	2024.00	2024.00	10332.33	11625.83
R_9^{20}	Time	616.20	4856.83	683.80	13088.48	158.60	104.00	215.80	156.00
	iter	2238.83	18819.17	2593.17	47889.33	408.67	425.83	715.00	983.83
R_{10}^{20}	Time	1149.21	4807.43	1245.41	16445.11	395.20	280.80	709.80	413.40
	iter	4005.50	19754.00	4769.50	72313.67	813.17	821.33	1894.00	2761.33
R_{11}^{20}	Time	2087.81	4518.83	2940.62	19697.73	1716.01	1846.01	3390.42	1630.21
	iter	6859.83	20004.00	11463.17	105411.50	2021.00	2024.00	7137.83	10201.50
R_{12}^{20}	Time	3226.62	4750.23	6981.04	23857.75	5181.83	7092.85	10358.47	5083.03
	iter	10536.00	20036.17	29638.83	146418.33	4015.33	4016.50	18918.83	27591.17
R_{13}^{20}	Time	1557.41	11398.47	1648.41	34289.02	353.60	213.20	517.40	348.40
	iter	3635.50	28862.00	4181.17	83259.83	606.17	614.33	1105.17	1723.17
R_{14}^{20}	Time	2688.42	11944.48	3359.22	42593.47	956.81	577.20	1788.81	930.81
	iter	5980.17	29744.67	8070.83	115804.67	1218.33	1238.50	3103.67	4558.17
R_{15}^{20}	Time	5525.04	10800.47	8759.46	47338.50	4565.63	4609.83	10088.06	3809.02
	iter	12124.33	29995.00	22847.50	164412.83	3019.67	3027.33	11770.83	15491.83
R_{16}^{20}	Time	5694.04	15358.30	13473.29	54015.35	11112.47	14734.29	20781.93	10059.46
	iter	13335.83	30041.83	38194.67	183127.33	6013.83	6014.83	25936.83	35517.50
AVG	Time	1520.36	4704.24	2745.62	16974.05	1673.92	1995.51	3341.35	1538.07
	iter	4635.79	16185.04	10612.04	71916.99	1477.38	1484.61	5767.46	7791.96

Table A.1: Execution time in ms and number of iterationson sprand networks with $|\mathcal{F}| = 20$.

		$SS\mathcal{M} - ES_s - P$	$SS\mathcal{M} - ES_s - nP$	$SS\mathcal{M} - FIFO - P$	$SS\mathcal{M} - FIFO - nP$	$\mathcal{N}S\mathcal{M} - ES_n - P$	$\mathcal{N}S\mathcal{M} - ES_n - nP$	$\mathcal{N}S\mathcal{M} - FIFO - P$	$\mathcal{N}S\mathcal{M} - FIFO - nP$
R_1^{60}	Time	70.20	382.20	88.40	855.41	18.20	46.80	23.40	36.40
	iter	881.83	4006.00	1292.17	8287.00	119.17	124.50	205.50	210.83
R_2^{60}	Time	80.60	421.20	124.80	1027.01	31.20	33.80	52.00	65.00
	iter	1181.00	4813.67	1783.00	13107.83	213.50	224.67	449.83	508.83
R_3^{60}	Time	148.20	345.80	314.60	1289.61	117.00	124.80	241.80	174.20
	iter	1847.83	5051.83	4688.50	22281.50	536.67	540.00	1649.83	1992.33
R_4^{60}	Time	231.40	306.80	829.41	1281.81	356.20	353.60	1128.41	374.40
	iter	3027.50	5108.83	13327.50	29710.67	1026.33	1026.33	5365.83	5407.67
R_5^{60}	Time	215.80	1505.41	301.60	3642.62	49.40	59.80	78.00	122.20
	iter	1547.83	8986.67	2348.50	20137.50	213.00	220.33	392.83	464.17
R_6^{60}	Time	338.00	1536.61	338.00	4206.83	114.40	101.40	184.60	189.80
	iter	2305.83	9777.33	2538.33	29493.67	423.33	428.33	950.67	1240.00
R_7^{60}	Time	561.60	1232.41	1105.01	5077.83	408.20	379.60	933.41	543.40
	iter	3717.67	10047.83	8623.50	47020.83	1026.83	1033.17	3564.67	4535.33
R_8^{60}	Time	574.60	1125.81	2722.22	5452.24	1469.01	1682.21	3497.02	1396.21
	iter	3994.17	10120.83	22441.67	63102.33	2028.00	2028.00	10479.17	11792.17
R_9^{60}	Time	730.60	5629.04	998.41	16907.91	179.40	163.80	351.00	270.40
	iter	2606.67	18858.83	3782.50	47892.17	412.17	426.50	853.83	982.00
R_{10}^{60}	Time	1302.61	5452.24	1627.61	21335.74	405.60	353.60	780.01	616.20
	iter	4338.17	19796.00	6154.67	72301.83	813.17	821.33	1947.17	2823.33
R_{11}^{60}	Time	3278.62	4786.63	4409.63	21301.94	2129.41	2051.41	4877.63	2033.21
	iter	9991.50	20065.33	17143.00	105675.17	2023.67	2024.00	8483.50	10366.33
R_{12}^{60}	Time	3832.42	4326.43	10322.07	25069.36	6510.44	7482.85	17006.71	5548.44
	iter	12258.33	20111.00	42805.67	146839.50	4023.17	4023.17	23720.83	27806.50
R_{13}^{60}	Time	2576.62	17729.51	2745.62	55369.96	426.40	273.00	803.41	517.40
	iter	5837.00	28901.67	6899.17	83314.50	611.83	616.33	1341.50	1640.50
R_{14}^{60}	Time	3361.82	17342.11	4739.83	63697.81	1190.81	722.80	2522.02	1281.81
	iter	7443.83	29788.17	11486.00	115917.50	1221.67	1238.50	3587.00	4623.17
R_{15}^{60}	Time	5735.64	17089.91	8990.86	59293.38	4583.83	4685.23	10171.27	4282.23
	iter	12398.50	30048.00	23353.00	164653.00	3021.83	3031.83	11899.50	15662.00
R_{16}^{60}	Time	5834.44	14378.09	14107.69	60039.58	11206.07	15168.50	20909.33	10467.67
	iter	13605.83	30110.17	40244.00	183389.67	6013.83	6014.83	26850.50	35689.33
AVG	Time	1804.57	5849.39	3360.36	21615.56	1824.72	2105.20	3972.50	1744.94
	iter	5436.47	15974.51	13056.95	72070.29	1483.01	1488.86	6358.89	7859.03

Table A.2: Execution time in ms and number of iterations on sprand networks with $|\mathcal{F}| = 60$.

		$SSM-ES_s-P$	$SSM-ES_s-nP$	$SSM-FIFO-P$	$SSM-FIFO-nP$	$\mathcal{N}SM-ES_n-P$	$\mathcal{N}SM-ES_n-nP$	$\mathcal{N}SM-FIFO-P$	$\mathcal{N}SM-FIFO-nP$
R_1^{100}	Time	83.20	494.00	91.00	1099.81	18.20	72.80	23.40	54.60
	iter	903.50	4010.00	1355.50	8289.00	121.00	125.00	203.33	208.83
R_2^{100}	Time	114.40	540.80	158.60	1242.81	39.00	46.80	59.80	93.60
	iter	1529.17	4855.17	2280.33	13159.50	219.83	228.67	456.67	516.33
R_3^{100}	Time	153.40	374.40	400.40	1372.81	124.80	132.60	304.20	244.40
	iter	2016.00	5107.33	6063.50	22436.17	536.17	540.67	1788.83	2074.17
R_4^{100}	Time	249.60	330.20	865.81	1479.41	403.00	418.60	1258.41	475.80
	iter	3144.17	5175.00	13488.83	29988.67	1034.83	1034.83	5499.67	5552.67
R_5^{100}	Time	241.80	1778.41	317.20	4201.63	49.40	98.80	88.40	171.60
	iter	1698.83	9014.17	2392.33	20153.33	217.33	224.17	396.17	473.17
R_6^{100}	Time	421.20	1648.41	483.60	4641.03	130.00	132.60	260.00	267.80
	iter	2800.50	9820.83	3701.50	29560.67	426.50	431.50	1066.33	1255.00
R_7^{100}	Time	600.60	1352.01	1146.61	5324.83	418.60	395.20	990.61	728.00
	iter	3895.00	10105.33	8798.17	47200.83	1027.83	1033.83	3661.17	4651.00
R_8^{100}	Time	624.00	1170.01	3062.82	5616.04	1885.01	1830.41	4173.03	1640.61
	iter	4275.33	10189.50	24648.00	63429.50	2035.33	2035.33	11191.17	11986.33
R_9^{100}	Time	865.81	6159.44	1211.61	18756.52	197.60	213.20	314.60	366.60
	iter	3064.67	18896.00	4571.83	47923.50	420.33	430.00	802.00	928.17
R_{10}^{100}	Time	1398.81	6086.64	1656.21	21863.54	413.40	426.40	832.01	821.61
	iter	4698.17	19839.00	6217.50	72388.33	813.17	821.33	2001.67	2881.00
R_{11}^{100}	Time	3447.62	4885.43	4565.63	21587.94	2171.01	2184.01	5202.63	2402.42
	iter	10330.17	20125.50	17546.50	105895.00	2023.67	2024.00	8583.83	10498.67
R_{12}^{100}	Time	3900.03	4459.03	15779.50	24650.76	9035.06	7659.65	29598.59	6086.64
	iter	12482.83	20181.50	61086.67	147221.83	4031.00	4031.00	27927.67	28026.67
R_{13}^{100}	Time	2652.02	17214.71	2945.82	53352.34	434.20	377.00	819.01	696.80
	iter	5837.00	28941.83	6923.50	83348.83	612.00	616.83	1363.17	1664.17
R_{14}^{100}	Time	3413.82	16114.90	4880.23	53968.55	1209.01	839.81	2639.02	1575.61
	iter	7532.33	29829.33	11566.50	115985.17	1229.33	1244.67	3509.50	4549.50
R_{15}^{100}	Time	8985.66	14521.09	11593.47	56547.76	5457.44	4986.83	15433.70	4706.03
	iter	17681.17	30103.00	29723.83	164930.00	3030.83	3032.83	14242.67	15847.67
R_{16}^{100}	Time	7407.45	12134.28	18857.92	50869.33	13083.28	15561.10	27781.18	11200.87
	iter	16266.17	30179.17	53216.67	183684.00	6017.33	6016.83	29899.67	35876.17
AVG	Time	2159.96	5578.99	4251.03	20410.94	2191.81	2210.99	5611.16	1970.81
	iter	6134.69	16023.29	15848.82	72224.65	1487.28	1491.97	7037.09	7936.84

Table A.3: Execution time in ms and number of iterations on sprand networks with $|\mathcal{F}| = 100$.

		$\mathcal{SSM}-ES_s-P$	$\mathcal{SSM}-ES_s-nP$	$\mathcal{SSM}-FIFO-P$	$\mathcal{SSM}-FIFO-nP$	$\mathcal{NSM}-ES_n-P$	$\mathcal{NSM}-ES_n-nP$	$\mathcal{NSM}-FIFO-P$	$\mathcal{NSM}-FIFO-nP$
R_1^{250}	Time	65.00	832.01	93.60	1856.41	33.80	117.00	28.60	130.00
	iter	903.50	4010.50	1355.50	8289.00	121.00	125.00	202.83	208.33
R_2^{250}	Time	156.00	780.01	236.60	1978.61	57.20	109.20	98.80	187.20
	iter	1983.67	4903.67	2991.50	13206.50	235.83	240.00	437.33	473.00
R_3^{250}	Time	247.00	527.80	527.80	2069.61	176.80	241.80	462.80	551.20
	iter	2839.00	5304.50	7020.33	23062.17	567.33	572.17	2016.33	2268.83
R_4^{250}	Time	327.60	418.60	1102.41	1939.61	590.20	616.20	1807.01	964.61
	iter	3829.50	5430.83	15787.33	30965.67	1056.33	1056.33	5988.00	6021.17
R_5^{250}	Time	301.60	2774.22	405.60	6448.04	65.00	192.40	106.60	371.80
	iter	2020.50	9059.83	3013.83	20157.50	218.67	230.33	380.00	445.17
R_6^{250}	Time	517.40	2290.61	764.41	6653.44	171.60	260.00	327.60	548.60
	iter	3260.33	9987.50	5400.50	29899.50	466.67	475.00	1025.33	1154.83
R_7^{250}	Time	951.61	1700.41	1508.01	6801.64	483.60	611.00	1586.01	1445.61
	iter	5599.17	10322.83	11020.67	47859.50	1034.83	1042.00	4336.67	5037.00
R_8^{250}	Time	777.41	1339.01	5519.84	6635.24	2321.81	2282.81	6289.44	2527.22
	iter	4973.00	10452.33	38132.50	64595.83	2057.33	2057.33	12469.17	12532.67
R_9^{250}	Time	1258.41	8632.06	1682.21	22009.14	234.00	416.00	403.00	761.81
	iter	4282.17	19049.83	6123.00	48069.67	434.67	441.67	795.83	913.17
R_{10}^{250}	Time	1768.01	7545.25	2306.21	25532.16	538.20	751.40	1292.21	1697.81
	iter	5633.83	19996.17	8388.50	72761.50	839.17	840.00	2434.67	3010.33
R_{11}^{250}	Time	4079.43	6068.44	6026.84	25628.36	2288.01	2800.22	5863.04	3809.02
	iter	11800.83	20333.33	22755.50	106665.50	2047.17	2057.00	8798.00	10528.83
R_{12}^{250}	Time	4425.23	5127.23	20040.93	26985.57	9770.86	8689.26	31800.80	8218.65
	iter	13107.67	20451.00	73686.33	148706.33	4058.67	4058.67	28720.67	28872.83
R_{13}^{250}	Time	3458.02	23756.35	3468.42	51516.73	595.40	722.80	1237.61	1502.81
	iter	7372.83	29084.33	8395.67	83572.67	629.33	625.33	1584.33	1733.50
R_{14}^{250}	Time	4547.43	19471.52	5132.43	56266.96	1245.41	1292.21	2992.62	3359.22
	iter	9567.33	29994.33	12295.33	116363.17	1234.83	1251.00	3700.00	4771.33
R_{15}^{250}	Time	10678.27	15810.70	13486.29	54881.15	5886.44	5831.84	17165.31	6892.64
	iter	18747.50	30320.33	34050.00	165756.83	3035.17	3037.17	14788.33	16348.50
R_{16}^{250}	Time	9575.86	14219.49	42419.27	53770.94	17942.72	17017.11	52356.54	13699.49
	iter	19513.83	30454.50	92384.00	184845.67	6021.83	6021.83	36476.50	36651.67
AVG	Time	2695.89	6955.86	6545.05	21935.85	2650.07	2621.95	7738.62	2916.73
	iter	7214.67	16197.24	21425.03	72798.56	1503.68	1508.18	7759.63	8185.70

Table A.4: Execution time in ms and number of iterations on sprand networks with $|\mathcal{F}| = 250$.

		$SSM - ES_s - P$	$SSM - ES_s - nP$	$SSM - FIFO - P$	$SSM - FIFO - nP$	$\mathcal{N}SM - ES_n - P$	$\mathcal{N}SM - ES_n - nP$	$\mathcal{N}SM - FIFO - P$	$\mathcal{N}SM - FIFO - nP$
R_1^{500}	Time	65.00	1495.01	106.60	3143.42	33.80	145.60	44.20	249.60
	iter	903.50	4010.50	1355.50	8289.00	121.00	125.00	202.83	208.33
R_2^{500}	Time	176.80	1245.41	249.60	3252.62	78.00	202.80	124.80	366.60
	iter	1983.67	4903.83	2991.83	13206.83	235.83	240.33	437.67	473.83
R_3^{500}	Time	369.20	821.61	725.40	3468.42	286.00	382.20	559.00	886.61
	iter	3642.33	5521.50	8588.17	23699.67	686.50	690.33	1747.17	1930.67
R_4^{500}	Time	478.40	595.40	1648.41	2779.42	886.61	899.61	2706.62	2152.81
	iter	4594.50	5881.83	18688.17	32886.50	1110.33	1110.33	6921.17	6974.83
R_5^{500}	Time	306.80	4695.63	447.20	10582.07	80.60	317.20	140.40	694.20
	iter	2020.50	9060.00	3013.83	20157.50	218.67	230.33	380.17	445.33
R_6^{500}	Time	676.00	4336.83	1092.01	10080.26	265.20	499.20	507.00	977.61
	iter	3947.83	10068.83	7072.50	30029.50	493.83	495.83	1032.67	1100.50
R_7^{500}	Time	1333.81	2633.82	2301.01	9731.86	842.41	1203.81	2459.62	2847.02
	iter	6840.50	10661.17	15242.67	48839.33	1180.00	1189.67	5017.33	5357.83
R_8^{500}	Time	1125.81	1879.81	7508.85	9573.26	3088.82	2971.82	8444.85	4443.43
	iter	6444.67	10878.83	40713.17	66656.50	2097.00	2097.00	13496.00	13588.17
R_9^{500}	Time	1908.41	15553.30	2225.61	37471.44	345.80	735.80	543.40	1362.41
	iter	5595.83	19100.83	7793.17	48108.17	441.00	447.50	765.00	853.50
R_{10}^{500}	Time	3060.22	11382.87	4737.23	41423.47	923.01	1287.01	2386.81	3107.02
	iter	8086.00	20269.33	15796.83	73207.00	902.33	903.67	2792.33	2951.33
R_{11}^{500}	Time	5600.44	8710.06	6687.24	39684.05	2483.02	3811.62	7251.45	6726.24
	iter	11800.83	20686.50	24359.67	108011.83	2058.33	2073.00	9580.33	11332.50
R_{12}^{500}	Time	6026.84	7560.85	25441.16	39910.26	11102.07	10948.67	37653.44	12617.88
	iter	14226.67	20898.67	80136.67	151245.67	4091.67	4091.67	30406.33	30648.33
R_{13}^{500}	Time	3941.63	35115.83	5194.83	93447.20	611.00	1177.81	1268.81	2545.42
	iter	7978.00	29284.50	12073.67	83814.17	651.17	649.67	1386.67	1533.83
R_{14}^{500}	Time	6177.64	29354.19	6479.24	101733.45	1536.61	2009.81	4357.63	5829.24
	iter	10735.33	30259.17	15223.67	116957.83	1271.50	1276.83	4119.83	4895.00
R_{15}^{500}	Time	16146.10	24489.56	20280.13	92490.39	6299.84	7324.25	21104.34	10883.67
	iter	22452.00	30685.83	47851.50	167219.17	3057.00	3057.67	16004.00	17078.33
R_{16}^{500}	Time	14458.69	19786.13	48680.11	74604.88	19663.93	19487.13	56173.36	18090.92
	iter	21800.83	30881.50	94777.33	186770.33	6028.50	6028.50	37768.17	37993.83
AVG	Time	3865.74	10603.52	8362.79	35836.03	3032.92	3337.77	9107.86	4611.29
	iter	8315.81	16440.80	24729.90	73693.69	1540.29	1544.21	8253.60	8585.39

Table A.5: Execution time in ms and number of iterations on sprand networks with $|\mathcal{F}| = 500$.

test		$\mathcal{D}$	$\mathcal{D}-P$	test	$\mathcal{D}$	$\mathcal{D}-P$	test	$\mathcal{D}$	$\mathcal{D}-P$	test	$\mathcal{D}$	$\mathcal{D}-P$	test	$\mathcal{D}$	$\mathcal{D}-P$
R_1^{20}	Time	31.20	23.40	R_1^{60}	93.60	26.00	R_1^{100}	114.40	31.20	R_1^{250}	150.80	41.60	R_1^{500}	296.40	52.00
	iter	208.17	186.17		212.60	205.50		208.83	203.33		208.33	202.83		208.33	202.83
R_2^{20}	Time	39.00	41.60	R_2^{60}	75.40	54.60	R_2^{100}	106.60	57.20	R_2^{250}	215.80	93.60	R_2^{500}	439.40	124.80
	iter	519.33	424.33		508.83	449.83		516.33	456.67		473.00	437.33		473.83	437.67
R_3^{20}	Time	122.20	174.20	R_3^{60}	197.60	247.00	R_3^{100}	296.40	309.40	R_3^{250}	707.20	496.60	R_3^{500}	1151.81	621.40
	iter	2069.83	1536.83		1992.33	1649.83		2074.17	1788.83		2268.83	2016.33		1930.67	1747.17
R_4^{20}	Time	293.80	1136.21	R_4^{60}	418.60	1154.41	R_4^{100}	587.60	1268.81	R_4^{250}	1266.21	1835.61	R_4^{500}	3101.82	3151.22
	iter	5256.00	5195.83		5407.67	5365.83		5552.67	5499.67		6021.17	5988.00		6974.83	6921.17
R_5^{20}	Time	57.20	52.00	R_5^{60}	135.20	85.80	R_5^{100}	197.60	98.80	R_5^{250}	426.40	117.00	R_5^{500}	826.81	143.00
	iter	459.67	330.00		464.17	392.83		473.17	396.17		445.17	380.00		445.33	380.17
R_6^{20}	Time	135.20	106.60	R_6^{60}	215.80	192.40	R_6^{100}	309.40	265.20	R_6^{250}	650.00	332.80	R_6^{500}	1167.41	535.60
	iter	1209.67	699.83		1240.00	950.67		1255.00	1066.33		1154.83	1025.33		1100.50	1032.67
R_7^{20}	Time	418.60	631.80	R_7^{60}	639.60	936.01	R_7^{100}	873.61	1014.01	R_7^{250}	1840.81	1648.41	R_7^{500}	3871.42	2652.02
	iter	4494.33	2991.83		4535.33	3564.67		4651.00	3661.17		5037.00	4336.67		5357.83	5017.33
R_8^{20}	Time	1219.41	3351.42	R_8^{60}	1476.81	3489.22	R_8^{100}	1783.61	4123.63	R_8^{250}	3211.02	6510.44	R_8^{500}	6323.24	8889.46
	iter	11625.83	10332.33		11792.17	10479.17		11986.33	11191.17		12532.67	12469.17		13588.17	13496.00
R_9^{20}	Time	184.60	218.40	R_9^{60}	312.00	345.80	R_9^{100}	418.60	330.20	R_9^{250}	878.81	400.40	R_9^{500}	1601.61	559.00
	iter	983.83	715.00		982.00	853.83		928.17	802.00		913.17	795.83		853.50	765.00
R_{10}^{20}	Time	452.40	715.01	R_{10}^{60}	702.00	772.20	R_{10}^{100}	969.81	800.81	R_{10}^{250}	2155.41	1302.61	R_{10}^{500}	4108.03	2542.82
	iter	2761.33	1894.00		2823.33	1947.17		2881.00	2001.67		3010.33	2434.67		2951.33	2792.33
R_{11}^{20}	Time	1773.21	3434.62	R_{11}^{60}	2311.42	4893.23	R_{11}^{100}	2808.02	5054.43	R_{11}^{250}	4810.03	5868.24	R_{11}^{500}	9170.26	7259.25
	iter	10201.50	7137.83		10366.33	8483.50		10498.67	8583.83		10528.83	8798.00		11332.50	9580.33
R_{12}^{20}	Time	5153.23	10379.27	R_{12}^{60}	5933.24	16894.91	R_{12}^{100}	6718.44	29666.19	R_{12}^{250}	9947.66	31397.80	R_{12}^{500}	17006.71	38009.64
	iter	27591.17	18918.83		27806.50	23720.83		28026.67	27927.67		28872.83	28720.67		30648.33	30406.33
R_{13}^{20}	Time	382.20	507.00	R_{13}^{60}	600.60	787.81	R_{13}^{100}	832.01	829.41	R_{13}^{250}	1861.61	1281.81	R_{13}^{500}	3073.22	1305.21
	iter	1723.17	1105.17		1640.50	1341.50		1664.17	1363.17		1733.50	1584.33		1533.83	1386.67
R_{14}^{20}	Time	1021.81	1804.41	R_{14}^{60}	1495.01	2516.82	R_{14}^{100}	1882.41	2477.82	R_{14}^{250}	3894.82	2844.42	R_{14}^{500}	7490.65	4061.23
	iter	4558.17	3103.67		4623.17	3587.00		4549.50	3509.50		4771.33	3700.00		4895.00	4119.83
R_{15}^{20}	Time	3772.62	9432.86	R_{15}^{60}	4469.43	9737.06	R_{15}^{100}	5184.43	14859.10	R_{15}^{250}	8239.45	16502.31	R_{15}^{500}	14175.29	20859.93
	iter	15491.83	11770.83		15662.00	11899.50		15847.67	14242.67		16348.50	14788.33		17078.33	16004.00
R_{16}^{20}	Time	9950.26	20646.73	R_{16}^{60}	10896.67	20664.93	R_{16}^{100}	11666.27	27461.38	R_{16}^{250}	15202.30	52889.54	R_{16}^{500}	21928.54	58287.17
	iter	35517.50	25936.83		35689.33	26850.50		35876.17	29899.67		36651.67	36476.50		37993.83	37768.17
AVG	Time	1562.93	3290.97	AVG	1873.31	3924.89	AVG	2171.83	5540.47	AVG	3466.15	7722.70	AVG	5983.29	9315.86
	iter	7791.96	5767.46		7859.14	6358.89		7936.84	7037.09		8185.70	7759.63		8585.39	8253.60

Table A.6: Execution time in ms and number of iterations for $\mathcal{D}$ and $\mathcal{D}-P$ on sprand networks.

		$SSM - ES_s - P$	$SSM - ES_s - nP$	$SSM - FIFO - P$	$SSM - FIFO - nP$	$NSM - ES_n - P$	$NSM - ES_n - nP$	$NSM - FIFO - P$	$NSM - FIFO - nP$
C_1^{20}	Time	374.40	8122.45	631.80	34403.42	101.40	111.80	213.20	187.20
	iter	1186.17	22374.17	2004.67	65410.83	162.00	163.50	363.83	479.83
C_1^{40}	Time	374.40	11018.87	629.20	32963.01	101.40	156.00	205.40	304.20
	iter	1190.00	22378.00	2013.00	65492.50	163.50	168.33	364.33	480.17
C_1^{60}	Time	382.20	15155.50	668.20	48141.91	109.20	210.60	228.80	457.60
	iter	1228.83	22388.67	2117.17	65836.83	170.83	171.33	367.83	488.17
C_1^{80}	Time	405.60	13223.68	722.80	55086.55	124.80	262.60	254.80	600.60
	iter	1300.00	22402.83	2288.33	65962.67	178.33	175.67	376.00	496.67
C_1^{100}	Time	423.80	12019.88	780.01	60811.79	145.60	358.80	273.00	738.40
	iter	1348.83	22415.83	2437.33	65985.50	184.67	184.33	386.00	503.50
C_2^{20}	Time	491.40	16211.10	722.81	59761.38	145.60	148.20	239.20	317.20
	iter	1097.67	32233.67	1631.67	95939.50	189.00	191.17	364.17	604.67
C_2^{40}	Time	520.00	30251.19	782.60	99916.04	161.20	247.00	306.80	525.20
	iter	1141.00	32248.67	1761.00	95999.67	195.83	198.50	385.83	618.83
C_2^{60}	Time	522.60	32630.21	798.21	147597.75	169.00	351.00	301.60	785.21
	iter	1143.83	32254.50	1780.83	96612.00	198.17	207.17	395.33	630.00
C_2^{80}	Time	499.20	29780.59	806.01	139860.10	171.60	429.00	296.40	967.21
	iter	1146.83	32257.50	1787.33	96680.17	199.33	209.33	398.33	627.33
C_2^{100}	Time	543.40	25656.96	819.01	122556.99	184.60	553.80	327.60	1167.41
	iter	1180.83	32264.50	1809.67	96785.83	204.17	213.33	409.50	631.00
C_3^{20}	Time	988.01	30048.39	1627.61	86835.36	223.60	184.60	442.00	351.00
	iter	1771.17	39814.17	2953.33	110732.00	214.67	215.50	456.00	596.00
C_3^{40}	Time	980.21	47234.50	1638.01	167363.07	234.00	288.60	447.20	574.60
	iter	1776.83	39819.83	2970.00	111007.33	218.83	222.17	462.50	598.83
C_3^{60}	Time	988.01	48063.91	1664.01	234680.10	247.00	418.60	457.60	813.81
	iter	1780.83	39823.83	2990.67	111446.00	220.50	225.00	467.83	604.00
C_3^{80}	Time	1001.01	46600.10	1658.81	226893.05	244.40	546.00	465.40	1034.81
	iter	1784.50	39827.50	2999.50	111584.50	223.50	229.67	472.83	607.17
C_3^{100}	Time	1014.01	43147.28	1677.01	223559.83	254.80	655.20	475.80	1266.21
	iter	1786.50	39829.50	3010.67	111853.50	224.33	233.17	475.50	609.50
C_4^{20}	Time	1279.21	80855.32	1528.81	267466.31	283.40	254.80	358.80	590.20
	iter	1489.83	62260.83	1777.17	147753.83	253.67	265.00	351.83	670.50
C_4^{40}	Time	1297.41	113321.73	1534.01	533630.02	280.80	429.00	366.60	912.61
	iter	1493.17	62264.17	1781.83	147759.17	254.17	270.50	354.33	673.00
C_4^{60}	Time	1310.41	120999.58	1518.41	698499.68	286.00	590.20	377.00	1263.61
	iter	1496.50	62267.50	1787.00	147847.33	255.00	273.67	356.00	674.00
C_4^{80}	Time	1310.41	121571.58	1518.41	667465.88	301.60	811.21	379.60	1593.81
	iter	1499.17	62270.17	1793.83	147978.83	257.83	279.17	359.83	676.83
C_4^{100}	Time	1302.61	131282.64	1526.21	724936.65	301.60	949.01	387.40	1931.81
	iter	1502.50	62273.50	1799.50	148109.17	260.00	280.67	362.83	678.50
AVG	Time	773.98	44521.73	1143.46	205657.49	198.42	368.79	337.73	760.57
	iter	1412.94	37968.38	2194.77	103094.65	208.97	215.68	398.46	593.25

Table A.7: Execution time in ms and number of iterations on complete networks.

		$\mathcal{D}$	$\mathcal{D}-P$
C_1^{20}	Time	369.20	210.60
	iter	479.83	363.83
C_1^{40}	Time	426.40	208.00
	iter	480.17	364.33
C_1^{60}	Time	587.60	223.60
	iter	488.17	367.83
C_1^{80}	Time	793.01	241.80
	iter	496.67	376.00
C_1^{100}	Time	990.61	270.40
	iter	503.50	386.00
C_2^{20}	Time	421.20	236.60
	iter	604.67	364.17
C_2^{40}	Time	738.40	280.80
	iter	618.83	385.83
C_2^{60}	Time	1040.01	291.20
	iter	630.00	395.33
C_2^{80}	Time	1318.21	304.20
	iter	627.33	398.33
C_2^{100}	Time	1609.41	330.20
	iter	631.00	409.50
C_3^{20}	Time	517.40	439.40
	iter	596.00	456.00
C_3^{40}	Time	829.41	452.40
	iter	598.83	462.50
C_3^{60}	Time	1068.61	475.80
	iter	604.00	467.83
C_3^{80}	Time	1365.01	462.80
	iter	607.17	472.83
C_3^{100}	Time	1656.21	473.20
	iter	609.50	475.50
C_4^{20}	Time	735.80	392.60
	iter	670.50	351.83
C_4^{40}	Time	1159.61	364.00
	iter	673.00	354.33
C_4^{60}	Time	1596.41	364.00
	iter	674.00	356.00
C_4^{80}	Time	2095.61	379.60
	iter	676.83	359.83
C_4^{100}	Time	2537.62	382.20
	iter	678.50	362.83
AVG	Time	1092.79	339.17
	iter	593.25	396.53

Table A.8: Execution time in ms and number of iterations for $\mathcal{D}$ and $\mathcal{D}-P$ on complete networks.

test		$\mathcal{EVD}$	$\mathcal{VD}$	test	$\mathcal{EVD}$	$\mathcal{VD}$	test	$\mathcal{EVD}$	$\mathcal{VD}$	test	$\mathcal{EVD}$	$\mathcal{VD}$	test	$\mathcal{EVD}$	$\mathcal{VD}$
R_1^{20}	cons	5602.21	5597.38	R_1^{60}	5616.74	5630.57	R_1^{100}	5646.54	5656.25	R_1^{250}	5675.59	5706.99	R_1^{500}	5774.44	5805.41
	sp	8.09	12.71		15.95	17.33		15.07	24.80		16.90	37.81		22.35	67.27
R_2^{20}	cons	5573.16	5574.84	R_2^{60}	5600.06	5610.18	R_2^{100}	5658.41	5674.12	R_2^{250}	5687.26	5724.80	R_2^{500}	5786.38	5823.55
	sp	8.04	11.71		12.05	20.72		23.45	36.08		25.14	54.33		33.47	96.70
R_3^{20}	cons	5600.32	5596.07	R_3^{60}	5606.27	5609.32	R_3^{100}	5618.97	5623.52	R_3^{250}	5698.09	5726.82	R_3^{500}	5869.13	5919.28
	sp	16.34	20.79		15.94	19.11		20.66	28.73		31.15	67.00		49.61	118.69
R_4^{20}	cons	5624.73	5623.89	R_4^{60}	5638.83	5638.86	R_4^{100}	5647.89	5652.71	R_4^{250}	5727.25	5743.98	R_4^{500}	5913.44	5937.00
	sp	22.15	23.68		24.13	27.11		24.88	39.07		43.08	67.25		66.27	124.12
R_5^{20}	cons	5635.30	5639.84	R_5^{60}	5699.81	5716.72	R_5^{100}	5795.05	5826.59	R_5^{250}	5864.79	5925.76	R_5^{500}	5996.08	6067.92
	sp	14.66	17.47		19.08	47.40		30.70	56.46		53.46	77.49		55.65	94.35
R_6^{20}	cons	5650.06	5649.02	R_6^{60}	5693.80	5697.76	R_6^{100}	5740.88	5727.65	R_6^{250}	5858.29	5908.13	R_6^{500}	6038.03	6103.38
	sp	16.07	20.23		21.95	30.36		29.49	50.26		44.64	75.41		56.13	137.12
R_7^{20}	cons	5738.89	5736.11	R_7^{60}	5756.82	5759.36	R_7^{100}	5778.20	5769.79	R_7^{250}	5860.86	5881.75	R_7^{500}	6071.35	6107.50
	sp	31.87	55.42		42.12	39.24		44.59	52.31		60.07	107.45		83.62	185.25
R_8^{20}	cons	5912.73	5905.47	R_8^{60}	5902.71	5905.32	R_8^{100}	5936.88	5926.18	R_8^{250}	5995.99	5996.25	R_8^{500}	6230.18	6245.62
	sp	102.06	99.37		109.53	119.84		118.52	163.57		125.39	166.84		185.34	245.06
R_9^{20}	cons	5819.24	5827.03	R_9^{60}	5910.01	5920.42	R_9^{100}	5967.62	5993.42	R_9^{250}	6271.68	6354.90	R_9^{500}	6615.10	6788.48
	sp	29.58	31.44		39.39	65.02		42.17	57.83		70.47	135.67		100.20	224.34
R_{10}^{20}	cons	5949.06	5954.40	R_{10}^{60}	5977.81	5972.60	R_{10}^{100}	6006.78	6009.14	R_{10}^{250}	6188.61	6209.15	R_{10}^{500}	6526.12	6632.76
	sp	39.73	44.78		50.09	68.26		50.03	73.34		94.09	161.70		130.46	260.54
R_{11}^{20}	cons	6321.15	6323.81	R_{11}^{60}	6331.72	6328.43	R_{11}^{100}	6352.36	6359.92	R_{11}^{250}	6420.05	6432.52	R_{11}^{500}	6659.34	6742.22
	sp	111.72	109.98		104.39	120.25		125.27	138.76		117.31	195.24		177.10	270.85
R_{12}^{20}	cons	6906.40	6903.58	R_{12}^{60}	6961.01	6953.37	R_{12}^{100}	6937.38	6929.36	R_{12}^{250}	7108.93	7101.29	R_{12}^{500}	7246.70	7262.48
	sp	372.29	325.60		385.10	367.18		351.46	393.41		397.94	489.02		437.26	569.90
R_{13}^{20}	cons	6113.50	6105.35	R_{13}^{60}	6165.31	6170.33	R_{13}^{100}	6266.11	6279.20	R_{13}^{250}	6513.68	6607.35	R_{13}^{500}	7038.76	7176.92
	sp	47.40	55.25		60.10	83.71		70.80	96.55		111.82	203.17		184.95	361.94
R_{14}^{20}	cons	6389.37	6387.83	R_{14}^{60}	6385.93	6380.82	R_{14}^{100}	6417.09	6427.59	R_{14}^{250}	6586.30	6583.40	R_{14}^{500}	7036.18	7108.97
	sp	79.35	83.09		78.60	83.80		86.36	104.65		120.02	201.04		190.23	340.72
R_{15}^{20}	cons	7267.52	7257.05	R_{15}^{60}	7265.00	7252.03	R_{15}^{100}	7194.80	7171.30	R_{15}^{250}	7384.54	7387.39	R_{15}^{500}	7696.73	7715.18
	sp	166.92	161.86		169.71	161.45		164.06	166.93		205.93	268.24		236.70	379.88
R_{16}^{20}	cons	9025.19	9017.37	R_{16}^{60}	8850.76	8835.18	R_{16}^{100}	8986.43	8983.50	R_{16}^{250}	8939.02	8934.19	R_{16}^{500}	9173.76	9194.15
	sp	580.41	660.94		577.78	689.43		580.36	700.09		601.24	754.36		671.03	803.64
AVG		6195.55	6193.69	AVG	6210.16	6211.33	AVG	6246.96	6250.64	AVG	6361.31	6389.04	AVG	6604.48	6664.43
		102.92	108.40		107.87	122.51		111.12	136.43		132.42	191.38		167.52	267.52

Table A.9: Average execution time for $\mathcal{EVD}$ and $\mathcal{VD}$ on sprand networks. In the rows cons we report the time to construct the anlarged graph, wherease the time needed to compute the shortest path is reported in rows sp.

		$\mathcal{EVD}$	$\mathcal{VD}$
C_1^{20}	cons	6042.90	6055.91
	sp	62.71	68.22
C_1^{40}	cons	6118.24	6128.56
	sp	73.48	88.99
C_1^{60}	cons	6225.25	6267.93
	sp	68.12	102.17
C_1^{80}	cons	6330.00	6348.94
	sp	78.90	109.12
C_1^{100}	cons	6406.58	6449.33
	sp	93.13	123.72
C_2^{20}	cons	6329.22	6349.24
	sp	93.95	115.42
C_2^{40}	cons	6440.55	6440.99
	sp	96.59	140.61
C_2^{60}	cons	6576.31	6630.63
	sp	101.78	159.23
C_2^{80}	cons	6747.59	6799.16
	sp	112.36	187.79
C_2^{100}	cons	6887.94	6960.69
	sp	138.19	197.74
C_3^{20}	cons	6607.07	6609.44
	sp	112.27	143.49
C_3^{40}	cons	6737.13	6745.34
	sp	112.24	154.03
C_3^{60}	cons	6832.94	6841.10
	sp	109.82	153.51
C_3^{80}	cons	6958.14	7017.74
	sp	112.41	158.85
C_3^{100}	cons	7085.43	7103.48
	sp	117.56	187.87
C_4^{20}	cons	7205.22	7236.52
	sp	151.31	161.64
C_4^{40}	cons	7602.76	7610.32
	sp	177.20	203.54
C_4^{60}	cons	7738.50	7756.33
	sp	177.03	224.01
C_4^{80}	cons	8006.01	8008.81
	sp	192.51	247.42
C_4^{100}	cons	8156.76	8227.06
	sp	198.02	258.03
AVG		6851.73	6879.38
		118.98	159.27

Table A.10: Average execution time for $\mathcal{EVD}$ and $\mathcal{VD}$ on complete networks.

		$\mathcal{NSM}-ES_n-P$		$\mathcal{EVD}$	$\mathcal{VD}$
N_1^{250}	time	7.88	cons	5820.90	5828.80
	iter	117.60	sp	27.33	76.64
N_1^{500}	time	63.24	cons	6004.00	6071.83
	iter	144.32	sp	51.17	119.17
N_1^{1000}	time	429.55	cons	6557.44	6743.04
	iter	211.70	sp	212.78	242.66
N_2^{250}	time	23.95	cons	6478.86	6707.73
	iter	102.46	sp	110.05	276.69
N_2^{500}	time	15.75	cons	7370.59	8356.27
	iter	107.50	sp	267.10	593.76
N_2^{1000}	time	23.43	cons	9654.96	11996.37
	iter	109.90	sp	622.17	1123.28
N_3^{250}	time	23.54	cons	7670.38	8234.57
	iter	101.63	sp	234.80	597.12
N_3^{500}	time	23.47	cons	9644.98	11906.09
	iter	104.67	sp	468.82	940.47
N_3^{1000}	time	39.87	cons	14230.95	22282.86
	iter	106.02	sp	1114.77	2077.98
N_4^{250}	time	86.71	cons	6352.99	6445.78
	iter	307.28	sp	100.13	256.79
N_4^{500}	time	102.08	cons	6937.41	7132.54
	iter	305.76	sp	205.48	535.07
N_4^{1000}	time	180.12	cons	7967.14	8641.46
	iter	315.94	sp	438.37	1133.55
N_5^{250}	time	273.53	cons	11468.26	11954.20
	iter	300.96	sp	501.63	983.04
N_5^{500}	time	289.04	cons	16011.61	18601.57
	iter	302.82	sp	1002.70	1883.75
N_5^{1000}	time	328.55	cons	25661.02	34755.77
	iter	303.72	sp	1944.67	3685.40
N_6^{250}	time	398.70	cons	21207.10	22849.30
	iter	300.10	sp	902.52	1794.93
N_6^{500}	time	452.82	cons	33474.80	42353.68
	iter	301.91	sp	1573.56	3111.10
N_6^{1000}	time	476.56	cons	59650.47	95068.25
	iter	301.42	sp	3473.21	7732.88
N_7^{250}	time	320.57	cons	7690.71	7754.17
	iter	504.34	sp	273.85	610.12
N_7^{500}	time	351.06	cons	8860.48	9432.68
	iter	508.22	sp	430.93	948.14
N_7^{1000}	time	437.56	cons	11550.40	13197.09
	iter	510.76	sp	1174.27	2324.04
N_8^{250}	time	1123.38	cons	27676.10	28849.34
	iter	500.77	sp	1223.12	2171.86
N_8^{500}	time	1201.21	cons	42825.22	47351.98
	iter	500.36	sp	2177.33	3868.60
N_8^{1000}	time	1319.21	cons	70658.80	90047.00
	iter	501.02	sp	3848.15	7456.45
N_9^{250}	time	1927.10	cons	68704.37	71807.87
	iter	500.01	sp	2175.06	3790.31
N_9^{500}	time	1888.16	cons	129045.98	130781.89
	iter	500.22	sp	3884.84	8598.41
N_9^{1000}	time	2106.15	cons	199622.89	283897.13
	iter	500.07	sp	7356.37	11865.42
AVG		515.30	AVG	30696.25	37742.57
		310.06		1325.75	2548.06

Table A.11: Results collected on fully random networks. For $\mathcal{NSM}-ES_n-P$ we report the computational time and the number of iterations, wherease for $\mathcal{EVD}$ and $\mathcal{VD}$ the time neede to construct the enlarged graph and that to compute the shortest path are given.

Chapter 4

Shortest path problem with forbidden paths: the elementary version [1]

Luigi Di Puglia Pugliese

Department of Electronics, Informatics and Systems, University of Calabria, 87036, Rende (CS), Italy

Francesca Guerriero

Department of Electronics, Informatics and Systems, University of Calabria, 87036, Rende (CS), Italy

Abstract

This paper addresses the elementary shortest path problem with forbidden paths. The main aim is to find the shortest paths from a single origin node to every other node of a directed graph, such that the solution does not contain any path, belonging to a given set (i.e., the forbidden set). It is imposed that no cycle can be included in the solution.

[1]Submitted for pubblication in the journal *European Journal of Operational Research.*

The problem at hand is formulated as a particular instance of the shortest path problem with resource constraints and two different solution approaches are defined and implemented. The former is a Branch & Bound based algorithm, the latter is a dynamic programming approach. Different versions of the proposed solution strategies are developed and tested on a large set of test problems.

Keywords: combinatorial optimization; shortest paths; forbidden paths; branch and bound approach; dynamic programming approach.

4.1 Introduction

The shortest path problem with forbidden paths (in the sequel referred to as $\mathcal{SPFP}$) has been introduced quite recently by Villeneuve and Desaulniers ([138]). It can be viewed as a variant of the constrained shortest path problem, in which the constraints are represented by a set of paths, that cannot be included in the solution.

The $\mathcal{SPFP}$ is defined on a directed graph $\mathcal{D} = (\mathcal{N}, \mathcal{A})$, where $\mathcal{N} = \{1, \dots, n\}$ denotes the set of nodes, and $\mathcal{A} \subseteq \mathcal{N} \times \mathcal{N}$ the set of arcs. It is assumed that the directed graph $\mathcal{D}$ is *simple*, that is no pair of nodes is connected by more than one arc.

A scalar cost c_{ij} is associated with each arc $(i, j) \in \mathcal{A}$. Given two distinct nodes o (referred to as origin node) and d (referred to as destination node), a *path* from node o to node d is a sequence of nodes $\Pi_{od}= \{o = i_1, \dots, i_q = d\}$, $q \geq 2$ and a corresponding sequence of $q-1$ arcs such that the h-th arc in the sequence is $(i_h, i_{h+1}) \in \mathcal{A}$ for $h = 1, \dots, q-1$. Thus, each path contains at least one arc. Since the directed graph is assumed to be simple, the sequence of nodes is sufficient to specify any path in $\mathcal{D}$. A path is said to be *elementary*, if it does not contain repeated nodes. An *oriented cycle* is a path for which the origin and destination nodes are the same. A digraph $\mathcal{D}$ is said to be d-connected if a path from node o to node d exists in $\mathcal{D}$.

The cost $z(\Pi_{od})$ of a path $\Pi_{od} = \{o = i_1, \ldots, i_q = d\}$ is defined as the sum of the costs of its arcs, that is $z(\Pi_{od}) = \sum_{h=1}^{q-1} c_{i_h, i_{h+1}}$. This path is said to be shortest/least-cost if it has minimum cost over all paths with the same origin and destination nodes. In the sequel, we assume that every oriented cycle in $\mathcal{D}$ has a non-negative cost. The forward star of node i (referred to as $FS(i)$) contains all the successor nodes of i, that is $FS(i) = \{j : (i,j) \in \mathcal{A}\}$.

Let F be a finite set of (forbidden) paths in $\mathcal{D}$. No assumptions are made about the characteristics of these paths. The $\mathcal{SPFP}$ consists of finding the least-cost path from node o to each other node $v \in N - \{o\}$, such that no paths in the solution contain any forbidden path $\pi \in F$.

The $\mathcal{SPFP}$ arises in all the contexts in which a shortest path must be modified, to exclude a set of paths, that do not satisfy a given feasibility criterion ([138]). In [138] a method, that combines solution techniques developed for the k shortest path problem ([100]) together with a method defined to solve the keyword matching problem ([2]), is proposed. The main idea is to build an enlarged graph, such that solving the shortest paths problem in this graph ensures that every path does not contain any forbidden path. Therefore, also the least-cost path in this new graph is feasible. It is allowed that the optimal paths contain repeated nodes. The $\mathcal{SPFP}$'s model can also be used to represent turn prohibitions in road networks. In this specific case, each forbidden path contains only two arcs and in the node shared by these arcs, a turn is prohibited. The shortest path problem with turn prohibition has been addressed recently in [78], where a labeling algorithm has been proposed. It is important to point out that also in this case a node can appear more than once in the optimal path.

It is worth observing that a feasible solution for the $\mathcal{SPFP}$ can contain repeated nodes, even if the graph does not present negative cost cycles. Indeed, the generation of a cycle allows to avoid that the built path solution contains a specific forbidden path (see Figure 4.1).

In this paper, we address the elementary version of the $\mathcal{SPFP}$ (referred to as $\mathcal{ESPFP}$). In particular, it is required that the optimal solution does not contain repeated nodes. To address the $\mathcal{ESPFP}$, two types of solution approaches, that operate on the original graph, are developed and tested. The former is of a Branch & Bound (B&B) type and it does not require that all the paths belonging to F are taken into account simultaneously. Indeed, the computational cost of this approach depends on the number of forbidden paths belonging to the optimal solution of the relaxed problem, in which the constraints, related to the forbidden paths, are not taken into account. The latter can be viewed as an extension of the dynamic programming approach proposed by Desrochers ([50]), to address the resource constrained shortest path problem, in which innovative node/label selection strategies are introduced.

The contribution of this paper is threefold. First of all, we address the elementary version of the $\mathcal{SPFP}$, that has not previously been considered in the scientific literature and, as mentioned in [138], seems to be of a different nature than the non elementary counterpart. Secondly, we give a mathematical formulation of the problem under study. Finally, solution approaches are defined and evaluated empirically. It is important to point out that the proposed methods are defined for solving the $\mathcal{ESPFP}$ in directed graphs with non-negative cost oriented cycles.

The paper is organized as follows. In the next Section we provide the mathematical formulation of the $\mathcal{ESPFP}$. Section 3 presents the solution approaches developed to solve the problem at hand. The behaviors of the proposed approaches are evaluated on a large set of test problems. The related computational results are reported in Section 4. Finally, Section 5 gives the conclusions of our work.

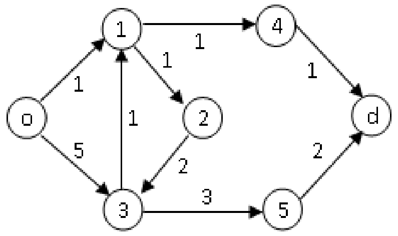

Figure 4.1: The cost c_{ij} is associated with each arc (i, j). The set of forbidden paths contains only one path $\pi =< o, 1, 4, d >$. The optimal path from node o to node d is $\Pi^*_{od} = \{o, 1, 2, 3, 1, 4, d\}$, with cost 7.

4.2 Modelling the $\mathcal{ESPFP}$

The $\mathcal{ESPFP}$ can be formulated as a particular instance of the resource constrained shortest path problem ([90]), as follows.

It is assumed that $|F|$ resources are available, one for each forbidden path, and we denote the resource limit vector by $\mathcal{W} = \{\mathcal{W}_1 = n_{\pi_1} - 2, \ldots, \mathcal{W}_{|F|} = n_{\pi_{|F|}} - 2\}$, where n_{π_k} represents the number of nodes of the forbidden path π_k, $k = 1, \ldots, |F|$.

In addition, let $w_{ij} = \{w^1_{ij}, \ldots, w^{|F|}_{ij}\}$ represent the resource consumed along the arc $(i, j) \in \mathcal{A}$, where $\forall \pi_k \in F$, $w^k_{ij} = 1$ if $(i, j) \in \pi_k$, 0 otherwise.

Given a path $\Pi_{od} = \{o = i_1, \ldots, i_l = d\}$ from the origin o to the destination d, the resource consumed along Π_{od} is $w(\Pi_{od}) = \{w^1(\Pi_{od}), \ldots, w^{|F|}(\Pi_{od})\}$, where $w^k(\Pi_{od}) = \sum_{(i,j)\in\Pi_{od}} w^k_{ij}$, $k = 1, \ldots |F|$.

From a mathematical standpoint, the single-source single-destination $\mathcal{ESPFP}$ (refered to as $\mathcal{ESPFP}^{od}$) can be represented by using the zero-one integer programming formulation, reported below:

$$\min \sum_{(i,j)\in\mathcal{A}} c_{ij}x_{ij} \tag{4.1}$$

$s.t.$

$$\sum_{\{j:(i,j)\in\mathcal{A}\}} x_{ij} - \sum_{\{j:(j,i)\in\mathcal{A}\}} x_{ji} = \begin{cases} 1 & \text{if } i = o \\ -1 & \text{if } i = d \\ 0 & otherwise \end{cases} \tag{4.2}$$

$$\forall i \in \mathcal{N}$$

$$\sum_{(i,j)\in\mathcal{A}} w_{ij}^k x_{ij} \leq \mathcal{W}_k, \quad \forall \pi_k \in F \tag{4.3}$$

$$x_{ij} \in \{0,1\} \quad \forall (i,j) \in \mathcal{A} \tag{4.4}$$

It is worth observing that, given the assumption on the oriented cycle cost and since it is not required to impose a lower limit on the resources consumed on the path chosen, the optimal solution of the mathematical model introduced above is an elementary path (for more details, the reader is referred to Beasley and Christofides [16]).

In order to give the mathematical formulation of the single-source all-destination version of the problem (i.e.: $\mathcal{ESPFP}$), let $\Phi^{(i)}, \forall i \in \mathcal{N} - \{o\}$ be the set of all possible elementary paths Π_{oi}^{ϕ}, $\phi = 1, \ldots, |\Phi^{(i)}|$ from node o to node i in the directed graph $\mathcal{D}$ and let x_{oi}^{ϕ}, $i \in \mathcal{N} - \{o\}$, $\phi = 1, \ldots, |\Phi^{(i)}|$ denote the binary variable indicating whether or not a path is selected. The $\mathcal{ESPFP}$ can be represented mathematically as follows:

$$\text{(4.5)} \qquad \min \sum_{i \in \mathcal{N}-\{o\}} z(\Pi_{oi}^{\phi}) x_{oi}^{\phi}$$

$$s.t.$$

$$\text{(4.6)} \qquad \sum_{\phi=1}^{|\Phi^{(i)}|} x_{oi}^{\phi} = 1, \quad \forall i \in \mathcal{N}-\{o\}$$

$$\text{(4.7)} \qquad w^k(\Pi_{oi}^{\phi}) x_{oi}^{\phi} \leq \mathcal{W}_k, \quad \forall \pi_k \in F, \forall i \in \mathcal{N}-\{o\}$$

$$\text{(4.8)} \qquad x_{oi}^{\phi} \in \{0,1\} \quad \forall i \in \mathcal{N}-\{o\}, \phi = 1, \ldots, |\Phi^{(i)}|$$

where constraints (4.6) ensure that for each node $i \in \mathcal{N}-\{o\}$, exactly one path is selected from the set $\Phi^{(i)}$, constraints (4.7) impose that for each node $i \in \mathcal{N}-\{o\}$, the chosen path $\Pi_{oi}^{\phi} \in \Phi^{(i)}$ does not contain any forbidden path. Finally, constraints (4.8) define the domain of the decision variables.

It is important to observe that the mathematical model reported above can be adopted to represent the single-source single-destination version of the $\mathcal{ESPFP}$, indeed, it is required to consider only the set $\Phi^{(d)}$.

4.3 Solution Approaches for the $\mathcal{ESPFP}$

In this Section, we provide a description of the methods proposed to address the problem under consideration. First, the B&B type approaches are presented, whereas in Section 4.3.2, the algorithms based on the dynamic programming are described.

4.3.1 Branch & Bound Approaches

In order to present the B&B approaches to solve the $\mathcal{ESPFP}$, we first address the single-origin single-destination version of the problem (i.e.,

$\mathcal{ESPFP}^{od}$). The proposed approach relies on the solution of a relaxation of $\mathcal{ESPFP}^{od}$ (referred to as $\mathcal{ESPFP}_R^{od}$), obtained by deleting the constraints (4.3). It is easy to verify that $\mathcal{ESPFP}_R^{od}$ reduces to the classical single-origin single-destination shortest path problem, that can be solved by applying any of the well-known methods, proposed in the scientific literature to solve this kind of problem ([20]).

If the optimal solution Π_R of $\mathcal{ESPFP}_R^{od}$ does not contain any forbidden path then it is also optimal for $\mathcal{ESPFP}^{od}$.

On the contrary, Π_R is not feasible and it contains at least one forbidden path. In this case, a set of problems are determined, starting from $\mathcal{ESPFP}_R^{od}$ by adding the violated constraints.

Branching Rule

In order to describe the branching procedure, let $\mathcal{V}$ be the set of arcs belonging to the forbidden paths, let $\mathcal{U}$ denote the remaining arcs and let $\mathcal{B}$ be the set given by the union of $\mathcal{V}$ and $\mathcal{U}$ (i.e., $\mathcal{B} = \mathcal{V} \cup \mathcal{U}$). The generic relaxed problem $\mathcal{ESPFP}_R^{od}$ is defined on the directed graph $\bar{\mathcal{D}} = (\mathcal{N}, \mathcal{B})$. At the first iteration, we set $\mathcal{B} \equiv \mathcal{A}$.

If the optimal path Π_R in $\bar{\mathcal{D}} = (\mathcal{N}, \mathcal{B})$ is not feasible for $\mathcal{ESPFP}^{od}$, because it contains at least one forbidden path, a set of problems (i.e., directed graphs $\bar{D}^\delta$) are determined starting from $\mathcal{ESPFP}_R^{od}$ (i.e., $\bar{\mathcal{D}} = (\mathcal{N}, \mathcal{B})$).

Assuming that only the $s-th$ forbidden path π_s is included in Π_R, $n_{\pi_s} - 1$ problems are generated during the branching phase. Let $(i_\delta, j_\delta)^s$ be the $\delta - th$ arc of the forbidden path π_s. The problems that are generated during the branching phase are defined on directed graphs, taking the following generic form $\bar{\mathcal{D}}^\delta(\mathcal{N}, \mathcal{B} - \{(i_\delta, j_\delta)^s\})$, $\delta = 1, \ldots, n_{\pi_s} - 1$. In other words, the directed graphs $\bar{\mathcal{D}}^\delta$, $\delta = 1, \ldots, n_{\pi_s} - 1$ are determined by removing from $\bar{\mathcal{D}} = (\mathcal{N}, \mathcal{B})$, the $\delta - th$ arc belonging to π_s. In this case $n_{\pi_s} - 1$ subproblems are generated; each of them is obtained by deleting, from the directed graph of the father node, one a time, the arcs belonging to π_s.

Let us now assume that $l > 1$ forbidden paths make infeasible the optimal path Π_R determined on the directed graph $\bar{\mathcal{D}}(\mathcal{N}, \mathcal{B})$ and let $\bar{F} = \{\pi_1, \pi_2, \ldots, \pi_l\}$ denote the set containing these paths. In this case, the directed graph $\bar{\mathcal{D}}(\mathcal{N}, \mathcal{B})$ is decomposed in $\delta = \sum_{i=1}^{l}(n_{\pi_i} - 1)$ subgraphs $\bar{\mathcal{D}}^{\delta}(\mathcal{N}, \bar{\mathcal{B}}^{\delta})$, where $\bar{\mathcal{B}}^{\delta} = \mathcal{B} - \{(i_h, j_h)^{\mu}\}$, $\mu = 1, 2, \ldots, l$ and $h = 1, 2, \ldots, n_{\pi_{\mu}} - 1$.

It is important to point out that, if the forbidden paths belonging to $\bar{F}$ share some arcs, the number of subproblems that are determined starting from $\bar{\mathcal{D}}(\mathcal{N}, \mathcal{B})$ is strictly less than δ. Consequently, δ represents an upper bound on the maximum number of subproblems that can be generated at each iteration of the proposed B&B method. A graphical representation of the proposed branching rule is depicted in Figure 4.2.

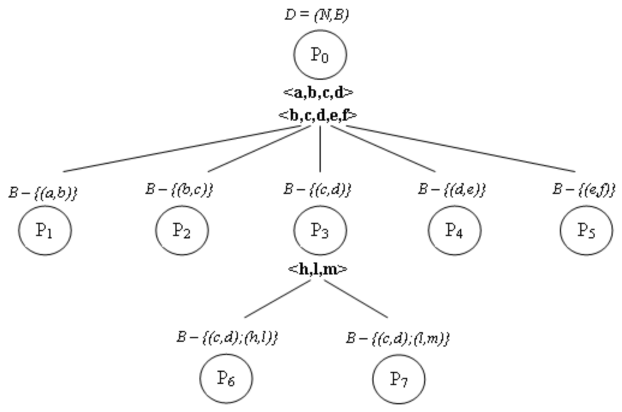

Figure 4.2: Graphical representation of the branching rule. For each node of the B&B tree we report in bold the forbidden paths that make infeasible the related relaxed problem, while in italic the set of arcs that define the problem associated with each node.

It is easy to verify that the proposed branching rule allows to eliminate the infeasible path that was optimal for the $\mathcal{ESPFP}_R^{od}$ without eliminating any feasible solutions. Indeed, the created subproblems are defined in such a way that every feasible path is feasible for at least one subproblem and the current infeasible solution is not feasible for any of the relaxed problems.

Upper Bounds for the $\mathcal{ESPFP}^{od}$

As for any B&B algorithm, the availability of a good upper bound can accelerate the search of the optimal solution. In order to compute a good quality upper bound for the $\mathcal{ESPFP}^{od}$, a heuristic procedure has been defined.

The proposed approach can be viewed as characterized by the execution of two main phases.

In the first phase, a directed graph $\bar{\mathcal{D}} = (\mathcal{N}, \bar{\mathcal{A}})$ is determined by adequately modifying the original directed graph $\mathcal{D} = (\mathcal{N}, \mathcal{A})$, whereas during the second phase the classical single-origin single-destination shortest path problem is solved in $\bar{\mathcal{D}} = (\mathcal{N}, \bar{\mathcal{A}})$. Let $\bar{\Pi}_{od}$ be the optimal path obtained as output of the second phase of the proposed heuristic procedure. We denote as $UB_d = z(\bar{\Pi}_{od})$ the upper bound on the optimal solution of the $\mathcal{ESPFP}^{od}$.

In particular, in the first phase, the set of arcs $\mathcal{A}$ is replaced by the set $\bar{\mathcal{A}} \subset \mathcal{A}$, that is defined in such a way that at least one arc of each of the forbidden paths does not belong to $\bar{A}$. This is obtained, by deleting from the set $\mathcal{A}$ one of the arcs belonging to each forbidden path π_r, $\forall \pi_r \in F$.

Let $\hat{F}$ represent the set of forbidden paths F ordered by increasing values of the number of arcs of the paths, that is $n_{\pi_l} - 1 \leq n_{\pi_{l+1}} - 1$, $l = 1, \ldots, |F| - 1$, the main operations executed by the developed heuristic to compute an initial upper bound for the $\mathcal{ESPFP}^{od}$ are depicted in Algorithm 5.

It is worth observing that, in the modified directed graph $\bar{\mathcal{D}}$, a path from node o to node d could not exist. This situation occurs when $\bar{\mathcal{D}}$ is not d-connected; in this case, $z(\bar{\Pi}_{od}) = +\infty$. However, it is possible to show that $\bar{\mathcal{D}}$ does not contain any of the forbidden paths belonging to F. Consequently, if $\bar{\mathcal{D}}$ is d-connected then the cost of the solution determined during the second phase of Algorithm 5 represents a valid upper bound for the problem under consideration.

Algorithm 5 Two phase heuristic procedure for computing an initial UB

```
**Step 0** *(Initialization)*
Set $\bar{\mathcal{A}} \equiv \mathcal{A}$; $UB_d = +\infty$.

**Step 1** *(Network Reduction)*
**repeat**
    Select the path $\pi_\mu$ at the top of $\hat{F}$.
    Select an arc $(i_\delta, j_\delta) \in \pi_\mu$
    Set $\bar{\mathcal{A}} = \bar{\mathcal{A}} - \{(i_\delta, j_\delta)\}$
    Set $\hat{F} = \hat{F} - \{\pi_\mu\}$
    Set $\hat{F} = \hat{F} - \{\pi_l : \pi_l \in \hat{F}, (i_\delta, j_\delta) \in \pi_l\}$
**until** $\hat{F} = \emptyset$

**Step 2** *(Shortest Path Computation)*
Solve the single-origin single-destination shortest path problem in the directed graph
$\bar{\mathcal{D}} = (\mathcal{N}, \bar{\mathcal{A}})$ obtaining the optimal path $\bar{\Pi}_{od}$.
Set $UB_d = z(\bar{\Pi}_{od})$.
```

In particular, the following theoretical results hold.

Theorem 4.3.1. ***Step 1*** *of Algorithm 5 provides a directed graph that does not contain any forbidden path.*

Proof. We will prove this theorem by contradiction. Thus, we assume (for the purposes of contradiction) that the directed graph $\bar{\mathcal{D}}$ determined by the proposed heuristic procedure, contains at least a forbidden path. In other words, it is assumed that there exists at least a forbidden path $\pi_\mu \in F$ such that the arc sequence $S_\mu = \{(i_h, i_{h+1}) \in \pi_\mu\}$, $h = 1, \ldots, n_{\pi_\mu} - 1$ belongs to $\bar{\mathcal{D}}$. It is important to observe, that this situation occurs only if $\forall (i,j) \in S_\mu$, $(i,j) \in \bar{\mathcal{A}}$. From the operations executed by the procedure, it is evident that for each forbidden path $\pi_\mu \in F$ at least one arc $(i,j) \in \pi_\mu$is removed from the set $\mathcal{A}$ and thus it does not belong to $\bar{\mathcal{A}}$. Consequently, only a subsequence of the arcs of the forbidden path π_μ can belong to $\bar{\mathcal{A}}$. This contradicts the assumption. $\square$

Search Strategy

The strategy for selecting the next subproblem to investigate determines how the B&B algorithm should proceed through the search tree. The order, in which the nodes are processed, determined by the search strategy, can have a significant effect on the behaviour of the algorithm.

In the proposed approach, in addition to the breath first search (BFS, for short) strategy, in which all nodes at one level of the search tree are processed before any node at a higher level, and the depth first search (DFS, for short) strategy, where the node with largest level in the search tree is chosen for the exploration, a best first search (BeFS, for shorts) strategy has been defined, which solves the most promising subproblem first.

For the BeFS, at each node is associated a label, that represents the cost of the solution of the father problem. In particular, let $\{SP_c^s\}$, $s \leq \delta$ denote the set of subproblems obtained by applying a branching rule to the candidate problem P_c and let $z(\Pi_{PR_c})$ be the cost of the optimal solution of the relaxed problem PR_c associated to P_c. To each problem SP_c^s is associated the label $E(SP_c^s) = z(\Pi_{PR_c})$. Let L be the set containing the problem to be solved. At each iteration, the subproblem $\tilde{P}_t$ selected from L is the one for which the minimum label value is obtained, that is $\tilde{P}_t = arg\ min\ \{E(P_c)|P_c \in L\}$.

It is worth observing that the minimum can be achieved for more than one subproblem. In this case, all the subproblems with the minimum label value, i.e., those belonging to the set $H = \{P_c \in L|E(P_c) = E_{min}\}$, where $E_{min} = min\{E(P_c)|P_c \in L\}$ are chosen and solved. It is important to point out that, if $|H| > 1$ then different strategies can be used to select the subproblem to be solved. In particular, two different selection rules have been considered, that is the $FIFO$ and the $LIFO$ approach.

The proposed method to address the $\mathcal{ESPFP}^{od}$ is formally stated in Algorithm 6.

Algorithm 6 B&B for the $\mathcal{ESPFP}^{od}$

```
STEP 0 (Initialization)
Set $\mathcal{B} = \mathcal{A}$, $L = \{P_0 \equiv \mathcal{D}(\mathcal{N}, \mathcal{B})\}$, $z = UB_d$.

STEP 1 (Problem selection)
Select and delete from $L$ a problem $P_c$, by applying a search strategy.

STEP 2 (Problem solution)
Solve the relaxed problem $PR_c$ associated to $P_c$.
Let $\Pi_{PR_c}$ denote the corresponding optimal solution.

STEP 3 (Feasibility Check and Branching)
if $\Pi_{PR_c}$ is unfeasible then
    if $z(\Pi_{PR_c}) < z$ then
        Generate the set of subproblems, by applying the proposed branching rule.
        Add each subproblem to $L$, if it does not already belong to it.
    end if
    Go to Step 4.
else
    if $z(\Pi_{PR_c}) < z$ then
        Set $z = z(\Pi_{PR_c})$.
        Go to Step 4.
    end if
end if

STEP 4 (Termination check)
if $L = \emptyset$ then
    STOP. $z$ is the optimal solution.
else
    Go to Step 1.
end if
```

In order to apply the BeFS strategy, the operations executed by this approach have to be slightly modified as follows. At **Step 0**, the set E, containing the label associated to the candidate subproblems, requires to be initialized. Indeed, we set $E(P_0) = 0$ and $E = \{E(P_0)\}$. At **Step 2**, the subproblem P_c to be solved is selected on the basis of the BeFS strategy. Finally, at **Step 4**, the label $E(SP_c^s) = z^*(PR_c)$ is associated to each new generated subproblem SP_c^s, $s \leq \delta$ and it is inserted in the set E.

It is worth observing that the label associated to each candidate subproblem represents a lower bound on the optimal cost and thus it can be used to fathom B&B nodes. In particular, at **Step 2**, the chosen subproblem P_c is discarded if $E(P_c) \geq z$. In other words, the subtree routed at the node corresponding to the relaxed problem PR_c is not explored.

Solving the $\mathcal{ESPFP}$

In this Section, the B&B approach developed to solve the $\mathcal{ESPFP}$ is described.

It is worth observing that the easiest way to address the $\mathcal{ESPFP}$ is to solve $|N|-1$ distinct $\mathcal{ESPFP}^{od}$, for which a different destination node d (i.e., $d \in \mathcal{N} - \{o\}$) is considered. This naive approach turns out to be very inefficient. Thus, the proposed solution procedure considers all the paths simultaneously and it can be viewed as a generalization of the B&B approach presented in the previous Sections.

Let $\mathcal{T}^*$ denote the shortest-path tree on the digraph $\mathcal{D} = (\mathcal{N}, \mathcal{B})$ and let Π^*_{oi} be the shortest path from node o to node i in $\mathcal{T}^*$, $\forall i \in N$. It is worth observing that the shortest-path tree $\mathcal{T}^*$ can be computed using any of the well-known label-correcting methods proposed in the scientific literature to address the single-origin all-destination shortest path problem. If at least a path Π^*_{oi} contains a forbidden path π_μ, the branching rule is applied generating a set of subproblems, defined on the digraphs $\bar{\mathcal{D}} = (\mathcal{N}, \bar{\mathcal{B}})$. Let $\bar{\mathcal{T}}$ be the solution on $\bar{\mathcal{D}} = (\mathcal{N}, \bar{\mathcal{B}})$ and let $\bar{\Pi}_{oi}$ denote the shortest path from node o to node i in $\bar{\mathcal{T}}$. Since $\bar{\mathcal{B}} \subset \mathcal{B}$, the following inequality holds:

$$z(\bar{\Pi}_{oi}) \geq z(\Pi^*_{oi}), \quad \forall i \in \mathcal{N}. \tag{4.9}$$

Consequently, if Π^*_{oi} is feasible, it represents the optimal path from

o to i, thus it is not required to find a path from o to i in the digraph $\bar{\mathcal{D}}$. On the other hand, if Π^*_{oi} is infeasible and $z(\Pi^*_{oi}) \geq z_i$, where z_i represents the cost of the best feasible path from o to i determined so far, by using condition (4.9) we have also that $z(\bar{\Pi}_{oi}) \geq z_i$, thus it is not necessary to consider the node i in the subsequent iterations. In other words, given the digraph $\bar{\mathcal{D}}$, it is required to solve the shortest paths problem from o to a subset of nodes $\bar{\mathcal{N}} \subset \mathcal{N}$, defined as $\bar{\mathcal{N}} = \mathcal{N} - \{i \in \mathcal{N} | i \neq o$ and Π^*_{oi} is feasible $\}$ $-\{i \in \mathcal{N} | i \neq o$ and Π^*_{oi} is infeasible and $\Pi^*_{oi} \geq z_i\}$.

For this reason, in what follows we use the notation $[\bar{\mathcal{D}}^c = (\mathcal{N}, \bar{\mathcal{B}}^c); \bar{N}^c]$ to represent a candidate problem P_c. Indeed, the relaxed problem PR_c consists of finding the shortest paths from the single origin node o to all the nodes belonging to $\bar{N}^c$ in the digraph $\bar{\mathcal{D}}^c = (\mathcal{N}, \bar{\mathcal{B}}^c)$.

Given a candidate problem P_c, the subproblems SP^c_s, $s \leq \delta$ are defined by using a strategy similar to that defined in the previous Section for the $\mathcal{ESPFP}^{od}$. In particular, let $\bar{F}^c$ be the set containing all the forbidden paths belonging to the optimal paths Π^*_{oi}, $i \in \bar{N}^c$. We assume that $|\bar{F}^c| = l$. The proposed branching rule allows us to define $\delta = \sum_{i=1}^{l} (n_{\pi_i} - 1)$ subproblems $\bar{D}^{c,\delta} = (N, \bar{B}^{\bar{c},\delta})$, where $\bar{B}^{\bar{c},\delta} = \bar{B}^c - \{(i_g, j_g)^\mu\}$, $\mu = 1, 2, \ldots, l$ and $g = 1, 2, \ldots, n_{\pi_\mu}$. If the forbidden paths belonging to $\bar{F}^c$ share some arcs, the number of subproblems that are determined starting from $\bar{\mathcal{D}}^c = (\mathcal{N}, \bar{\mathcal{B}}^c)$ is strictly less than δ.

The proposed method to address the $\mathcal{ESPFP}$ is formally stated in Algorithm 7.

It is important to point out that also in this case, by using a two phases heuristic approach similar to the one described in Section 4.3.1, it is possible to determine an upper bound for the $\mathcal{ESPFP}$. In particular, let $gri(i)$ and $gro(i)$ denote the number of ingoing and outgoing arcs of node i, the **Step 1** is modified as reported in Algorithm **8**, whereas in the **Step 2**, the shortest path problem from the origin node o to all other nodes is solved on the digraph obtained from **Step 1**. For each node $i \in \mathcal{N}$, a path $\bar{\Pi}_{oi}$ is determined and the related cost

Algorithm 7 B&B for the $\mathcal{ESPFP}$

```
Step 0 (Initialization Phase)
Set $L = \{P_0 \equiv [\mathcal{D}^0(\mathcal{N}, \mathcal{B} \equiv \mathcal{A}); \bar{\mathcal{N}}^0]\}$;
Set $z_j = UB_j, \forall i \in \mathcal{N} - \{o\}; z_o = 0$.

Step 1 (Problem selection)
Select and remove from $L$ a candidate problem $P_c \equiv [\mathcal{D}^c = (\bar{\mathcal{N}}, \bar{\mathcal{A}}^c); \bar{\mathcal{N}}^c]$, by applying a
search strategy.

Step 2 (Problem solution)
Solve the relaxed problem $PR_c$;
Let $\mathcal{T}_c^*$ denote the optimal solution of $PR_c$ and let $z_j^*(PR_c)$ be the cost of the optimal
path $\Pi_{oj}^c$ in $\mathcal{T}_c^*$ from node $o$ to node $j$, $\forall j \in \bar{\mathcal{N}}^c$.

Step 3 (Feasibility check and Branching)
Set $\bar{F}^c = \emptyset$.
for all $j \in \bar{\mathcal{N}}^c$ do
  if $\Pi_{oj}^c$ is infeasible then
    if $z_j^*(\Pi_{oj}^c) < z_j$ then
      Set $\bar{F}^c = \bar{F}^c \cup \{f_\mu\}$, where $f_\mu$ makes unfeasible $\Pi_{oj}^c$.
    end if
  else
    if $z_j^*(\Pi_{oj}^c) < z_j$ then
      Set $z_j = z_j^*(\Pi_{oj}^c)$
    end if
  end if
end for
if $\bar{F}^c \neq \emptyset$ then
  By applying the proposed branching rule, let generate a set of subproblems $\{SP_s^c\}, s \leq$
  $\delta$.
  Add a subproblem $SP_s^c$ to $L$ if it does not already belong to $L$.
end if
Go to Step 1.

Step 4 (Termination check)
if $L = \emptyset$ then
  STOP. $z_j$ is the cost of the optimal path from node $o$ to node $j$, $\forall j \in \mathcal{N}$.
else
  Go to Step 1.
end if
```

is used to define an upper bound on the optimal solution of $\mathcal{ESPFP}$.

In particular, $z(\bar{\Pi}_{oi})$ is used to initialize the upper bound for each node i, that is $UB_i = z(\bar{\Pi}_{oi})$.

Algorithm 8 Step 1 of the heuristic procedure for computing an initial UB for $\mathcal{ESPFP}$

```
**Step 1** *(Network reduction)*
**repeat**
  Select the path $\pi_\mu$ at the top of $\hat{F}$.
  Select an arc $(i_\delta, j_\delta) \in \pi_\mu$ such that
      in the case of $i_\delta = o$, $gro(i_\delta) > 1$ and $gri(j_\delta) > 1$
      in the case of $i_\delta \neq o$, $gri(j_\delta) > 1$
  **if** Such an arc does not exist **then**
    STOP The problem is infeasible
  **else**
    Set $\bar{\mathcal{A}} = \bar{\mathcal{A}} - \{(i_\delta, j_\delta)\}$
    Set $\hat{F} = \hat{F} - \{\pi_\mu\}$
    Set $\hat{F} = \hat{F} - \{\pi_l : \pi_l \in \hat{F}, (i_\delta, j_\delta) \in \pi_l\}$
    Set $gri(j_\delta) = gri(j_\delta) - 1$ and $gro(i_\delta) = gro(i_\delta) - 1$
  **end if**
**until** $\hat{F} = \emptyset$
```

It is worth observing that the proposed heuristic checks also the feasibility of the problem. In other words, if the STOP condition (see Algorithm **8**) is verified, then the problem is infeasible. On the other hand, if the STOP condition is not verified, then the digraph $\bar{\mathcal{D}} = (\mathcal{N}, \bar{\mathcal{A}})$ is connected, that is for each node $i \in \mathcal{N} - \{o\}$, $gri(i) > 1$ and $gro(o) > 1$. Thus, the **Step 2** provides a feasible solution for the $\mathcal{ESPFP}$.

All the search strategies and the fathoming rule defined for the $\mathcal{ESPFP}^{od}$ have been extended to the single-origin all-destination version of the problem. More specifically, in order to fathom unpromising nodes, a vector $E(SP^c) \in R^{|\mathcal{N}|}$, whose generic element $E(SP^c)_j$ is defined as $E(SP^c)_j = z_j^*(PR_c)$ is associated to each subproblem SP^c. As a consequence, a problem P_c, belonging to L, is discarded only if for all $j \in \bar{\mathcal{N}}^c$, the condition $E(P_c)_j \geq z_j$ holds. In addition, in the $BeFS$ strategy, the problem to be solved is determined on the basis of the value $e(P_c) = \max_{j \in \mathcal{N}}\{E(P_c)_j\}$.

Let $\mathcal{O}(\alpha)$ be the complexity of the algorithm used to solve the relaxed problems and let m denote the average number of arcs belonging to the forbidden paths (i.e.: $m = \frac{\sum_{\pi \in F} n_\pi - 1}{|F|}$), the following theoretical results hold.

Lemma 4.3.2. *In the worst case, the complexity of the proposed B&B approach is* $\mathcal{O}\left(\alpha m^{log_m(m|F|+1)}\right)$.

Proof. In the worst case, each problem of each level of the B&B tree could be infeasible, because at least one constraint of type (4.3) is not satisfied. This means that for each problem, the branching procedure must be executed. Let r denote the maximum number of levels of the B&B tree. The maximum number of problems generated and solved is m^r. The total number of arcs that belong to the forbidden paths is an upper bound on the number of problems generated and solved, that is $m^r \leq (m\,|F|+1)$. In the worst case, we have $m^r = (m\,|F|+1)$. From this relation, it is possible to derive the number of levels that is $r = log_m(m\,|F|+1)$. Thus, the maximum number of nodes is equal to $m^{log_m(m|F|+1)}$. Since $\mathcal{O}(\alpha)$ represents the computational complexity of the approach used to solve each of the relaxed problems, we have that the complexity of the proposed method is equal to $\mathcal{O}\left(\alpha m^{log_m(m|F|+1)}\right)$. □

4.3.2 Dynamic Programming Approaches

Dynamic programming approaches, that can be viewed as an extension of the algorithm proposed by Desrochers ([50]), are developed to find the optimal solution of the $\mathcal{ESPFP}$.

In order to describe the proposed solution methods, it is useful to introduce the following notations and definitions.

Let $l_j = (z_j, w_j)$ be a label, representing a path from node o to node j. In particular, $z_j = z(\Pi_{oj})$ represents the cost of the path, whereas w_j is the vector containing information about the resources consumed along Π_{oj}, that is $w_j = w(\Pi_{oj})$. It is worth observing that more than one path Π_{oj} may exist to reach node j, starting from node o. This means that several labels $l_j^\xi = (z_j^\xi, w_j^\xi)$, can be associated with each node j and they are stored in the set $D(j)$. In particular, $l_j^\xi = (z_j^\xi, w_j^\xi)$, $\xi = 1, \dots, |D(j)|$ stores information related to the $\xi - th$ path Π_{oj}^ξ from node o to node j. Since each label is associated with a partial path, in what follows the terms "label" and "solution" are used in an interchangeable way.

Definition 4.3.3. *Let l_j^ξ and l_j^ζ be two labels associated with the node j, we say that l_j^ξ dominates l_j^ζ if the following conditions hold*

$$z_j^\xi \le z_j^\zeta; \tag{4.10}$$

$$w_j^\xi \le w_j^\zeta; \tag{4.11}$$

and at least one of the inequalities is strict.

Definition 4.3.4. *Two labels l_j^ξ and l_j^ζ associated with the same node j are said to be equivalant if $z_j^\xi = z_j^\zeta$ and $w_j^\xi = w_j^\zeta$.*

Definition 4.3.5. *A solution is efficient if it is not equivalent to each other solutions and does not exist another solution that dominates it.*

Definition 4.3.6. *The $\xi - th$ label $l_j^\xi = (z_j^\xi, w_j^\xi)$, associated with the node j is referred to as a feasible solution if and only if $w^k(\Pi_{oj}^\xi) \le \mathcal{W}_k$, $k = 1, \dots, |F|$.*

Starting from the label $l_o = (0, 0)$, associated with the origin node o, the solution space is explored in order to obtain efficient and feasible solutions for each node. At the end of the algorithm, the set $D(j), \forall j \in \mathcal{N}$ is an efficient set, that is it contains efficient and feasible solutions. During the course of the algorithm, a label $l_j^\xi = (z_j^\xi, w_j^\xi)$ is generated starting from a label $l_i^\zeta = (z_i^\zeta, w_i^\zeta)$, using the following updating rules:

$$z_j^\xi = z_i^\zeta + c_{ij}. \tag{4.12}$$

$$w^k(\Pi_{oj}^\xi) = w^k(\Pi_{oi}^\zeta) + w_{ij}^k, \quad k = 1, \dots |F|\,. \tag{4.13}$$

A new label is created only if it is feasible (see Definition 4.3.6).

For each node $j \in \mathcal{N}$, $z^*_j = z(\Pi^*_{oj}) = \min_{l^\xi_j \in D(j)}\{z^\xi_j\}$ represents the value of the optimal path, without forbidden paths, from node o to node j.

It is important to point out that all the labels associated with paths that contain a cycle of length zero are discarded. For the sake of semplicity and without loss of generality, let us suppose that a cycle $cyc =< i, \ldots, i >$ at node i exists. Let $\check{\Pi}_{oi}$ be a path, without cycles, from node o to node i and let $\hat{\Pi}_{oi} = \check{\Pi}_{oi} \cup cyc$ be a path that contains the cycle cyc. It is easy to verify that $z(\hat{\Pi}_{oi}) = z(\check{\Pi}_{oi})$ and $w(\hat{\Pi}_{oi}) \geq w(\check{\Pi}_{oi})$. Thus the label associated with path $\hat{\Pi}_{oi}$ is either dominated by $(w^k(\hat{\Pi}_{oi}) > w^k(\check{\Pi}_{oi})$, for some $k = 1, \ldots, |F|)$ or equivalent to $(w^k(\hat{\Pi}_{oi}) = w^k(\check{\Pi}_{oi}), \;\; \forall k)$ the label associated with path $\check{\Pi}_{oi}$. Since, from Definition 4.3.5, the label associated with the path $\hat{\Pi}_{oi}$ is not efficient, it is discarded. As a consequence, only paths without cycles are determined by the proposed methods.

The solution approaches developed to address the $\mathcal{ESPFP}$ have been defined by considering different label and node selection strategies. In the first case, at each iteration, a label, associated with a node i is selected and a new label is determined, extending the former, for each $j \in FS(i)$. In a node selection method, at each iteration, a node i is selected and all the labels associated with it are used to updated the set of labels of the successor nodes of i.

In what follows, we describe the proposed label selection ($\mathcal{LSM}$ for short) and node selection methods (refered to as $\mathcal{NSM}$). The developped approaches mantain a candidate list L that, in the case of $\mathcal{LSM}$, stores all the labels with the potential to determine a new label for at least one node. In the case of $\mathcal{NSM}$, the list L contains the node to be processed.

The strategy used to select, at each iteration, the label/node to be treated influences the efficiency of the defined approaches. In this

Algorithm 9 $\mathcal{LSM}$ scheme

```
Step 0 (Initialization phase)
Set $D(o) = \{l_o^1\}; l_o^1 = (0,0); D(j) = \emptyset, \forall j \in \mathcal{N} - \{o\}; L = \{l_o^1\}$.

Step 1 (Label selection)
Select and remove from $L$ a label $l_i^\zeta = (z_i^\zeta, w_i^\zeta)$.

Step 2 (Label scanning)
for all $j \in FS(i), j \neq o$ do
    Compute $\bar{w}_j$ by using condition (4.13).
    if the newly created label is feasible then
        Set $\bar{z}_j = z_i^\xi + c_{ij}$
        if $(\bar{z}_j, \bar{w}_j)$ is dominated by some label belonging to $D(j)$ then
            Go to Step 1.
        else
            Set $l_j^{|D(j)|+1} = (\bar{z}_j, \bar{w}_j); D(j) = D(j) \cup \{l_j^{|D(j)|+1}\}; L = L \cup \{l_j^{|D(j)|+1}\}$.
            Remove from $D(j)$ and $L$ all the labels that are dominated by $l_j^{|D(j)|+1}$.
        end if
    end if
end for
Go to Step 1.

Step 4 (Termination check)
if $L = \emptyset$ then
    STOP.
else
    Go to Step 1.
end if
```

work we consider classical selection strategies, that is the Bellman-Ford rule (BF, for short) in which the list L is organized as a FIFO queue and the $LIFO$ strategy in which the list L is managed as a stack, that is a label/node is selected from and added to the top of L.

Other selection strategies have been considered that are specialized for both $\mathcal{LSM}$ and $\mathcal{NSM}$. In the next Sections we provide a detailed description of the proposed rules.

Label Selection Method

At each iteration of $\mathcal{LSM}$, a label l_i^ζ is selected to be treated. Efficient and feasible labels are generated and stored.

The steps of the proposed algorithm are depicted in Algorithm 9. It is possible to show that at the end of the algorithm the set of all efficient and feasible solutions is available at each node.

Proposition 4.3.7.

1. *At the end of each iteration, the following conditions holds:*
   (a) $D(o) = \{l_o^1\} = \{(0,0)\}$*;*
   (b) $\forall j \in \mathcal{N}$*, if* $D(j) \neq \emptyset$ *[i.e.:* $D(j) = \{l_j^1, l_j^2, \dots, l_j^l\}$*] and* $j \neq o$*, then* $l_j^\xi, \xi = 1, \dots, l$ *is a label related to a feasible path from node* o *to node* j *and* $D(j)$ *is an efficient set.*
2. *Upon termination of the algorithm, if* $D(j) \neq \emptyset, j \in \mathcal{N}$ *and* $j \neq o$*, then* $D(j)$ *contains the labels of all efficient and feasible paths from node* o *to node* j*.*

Proof. 1. Condition 1a holds because, initially, $D(o) = \{l_o^1\} = \{(0,0)\}$ and, by the rules of the algorithm, $D(o)$ cannot change.

We prove condition 1b by induction on the iteration count. Indeed, initially, condition 1b holds, since node o is the only node for which the set $D(o)$ is nonempty. Suppose that 1b holds for some node j at the beginning of some iteration.

Let l_i^ζ be the label removed from L.

If $i = o$, it happens only at the first iteration and $\zeta = 1$. At the end of this iteration, we have $D(j) = l_j^1$ for all successor nodes j of o such that the corresponding path Π_{oj} is feasible, $D(j) = \emptyset$ for all other nodes $j \neq o$, $j \notin FS(i)$. Thus, the set of labels has the required property.

If $i \neq o$, then l_i^ζ is the label of some feasible path Π_{oi}^ζ starting from o and ending to i that is not dominated by the other paths in $D(i)$, by the induction hypothesis. If the set $D(j)$ changes, for some node j, such that $j \in FS(i)$, as a result of the iteration, a new feasible label $\bar{l}_j = (\bar{z}_j, \bar{w}_j)$ is obtained for node j. The created label is related to the feasible path Π_{oj} consisting of path Π^{ζ}_{oi} followed by the arc (i, j). Finally, note that, by the rules of the algorithm, the newly created label is added to $D(j)$ only if it is an efficient label. This completes the induction proof of 1b.

2. Using part 1b, we have that, at each iteration, $\forall j \in \mathcal{N}$ such that $D(j) \neq \emptyset$, $D(j)$ is an efficient set. Thus, the property mentioned is also satisfied when the algorithm terminates. In addition, the way in which the candidate list L is updated and the termination condition (i.e., the algorithm terminates when there are no more labels left to be scanned) guarantee that all the labels with the potential to determine a new label for at least one node are scanned during the execution of the algorithm.

$\square$

In what follows, we report a description of the strategies used to select at each iteration the label to be treated and to manage the list L.

DP This strategy can be viewed as an extension of the D'Esopo Pape approach [115]. At each iteration, the label at the top of L is selected. In addition, a label is inserted at the bottom of the list L if it represents the first label of the corresponding node, otherwise it is inserted at the top.

SLF The label at the top of L is extracted, whereas a label is inserted in the list L by following the Small Label First rule. In particular, let l^{ξ}_i be the label at the top of L and let l^{ζ}_j denote the label to be inserted, *SLF* executes the following operations:

if $z^{\xi}_j \leq z^{\zeta}_i$ **then**

Insert l^{ξ}_j at the top of L.

```
else
Insert $l_j^\xi$ at the bottom of $L$.
end if
```

LLL This is the Large Label Last (i.e. LLL) strategy, described in what follows. Let l_i^ζ be the label at the top of L and let $s = \frac{\sum_{l_j^\xi \in L} z_j^\xi}{|L|}$, LLL executes the operations reported below:

```
repeat
Move $l_i^\zeta$ to the bottom of $L$.
until $z_i^\zeta \leq s$
Select the label at the top of $L$.
```

SLS The selected label l_i^ζ is the one for which the following condition is satisfied $l_i^\zeta = argmin_{l_j^\xi \in L} \left\{ z_j^\xi \right\}$.

$LLL - SLF$ This rule is a hybrid strategy, that has been defined combining the selection strategy LLL with the insertion rule SLF.

Node Selection Method

In the $\mathcal{NSM}$, at each iteration a node i is selected and efficient and feasible labels are generated from the labels belonging to $D(i)$ that are not yet extended. In Algorithm 10, we present the steps of the proposed $\mathcal{NSM}$. Also in this case, it is possible to show that at the end of the algorithm the set of efficient solutions are available at each node.

Proposition 4.3.8.

1. *At the end of each iteration, the following conditions holds:*

   (a) $D(o) = \{l_o^1\} = \{(0,0)\}$*;*

Algorithm 10 $\mathcal{NSM}$ scheme

```
Step 0 (Initialization phase)
Set $D(o) = \{l_o^1\}; l_o^1 = (0,0); D(j) = \emptyset, \forall j \in \mathcal{N} - \{o\}; L = \{o\}$.

Step 1 (Node selection)
Select and delate from $L$ a node $i$.

Step 2 (Node processing)
for all $j \in FS(i), j \neq o$ do
    for all $l_i^\zeta \in D(i)$ not yet extended do
        Compute $\bar{w}_j$ by using condition (4.13).
        if the newly created labels is feasible then
            Set $\bar{z}_j = z_i^\xi + c_{ij}$
            if $(\bar{z}_j, \bar{w}_j)$ is dominated by some label belonging to $D(j)$ then
                Go to Step 1.
            else
                Set $l_j^{|D(j)|+1} = (\bar{z}_j, \bar{w}_j); D(j) = D(j) \cup \{l_j^{|D(j)|+1}\}$.
                Remove from $D(j)$ all the labels that are dominated by $l_j^{|D(j)|+1}$.
                Add $j$ to $L$ if it does not already belong to it.
            end if
        end if
    end for
end for
Go to Step 1.

Step 3 (Termination check)
if $L = \emptyset$ then
    STOP.
else
    Go to Step 1.
end if
```

(b) *$\forall j \in \mathcal{N}$, if $D(j) \neq \emptyset$ [i.e.: $D(j) = \{l_j^1, l_j^2, \ldots, l_j^l\}$] and $j \neq o$, then $l_j^\xi, \xi = 1, \ldots, l$ is a label related to a feasible path from node o to node j and $D(j)$ is an efficient set.*

2. *Upon termination of the algorithm, if $D(j) \neq \emptyset, j \in \mathcal{N}$ and $j \neq o$, then $D(j)$ contains the labels of all efficient and feasible paths from node o to node j.*

Proof. 1. The proof of condition 1a is identical to the corresponding part of Proposition 1.

We prove condition 1b by induction on the iteration count. Initially, condition 1b holds, since node o is the only node for which the set $D(o)$ is nonempty. Suppose that 1b holds for some node j at the start of some iteration at which the node removed from L is i. The following alternative situations can occur:

- $i = o$. This situation happens only at the first iteration and at the end of the iteration we have

$$D(j) = \begin{cases} l_j^1 = (z_j^1, w_j^1), & \text{if } w_j^1 \leq \mathcal{W} \text{ and } j \in FS(i) \\ \emptyset, & \text{otherwise} \end{cases}$$

  Thus, the set of labels $D(j)$ has the required property.

- $i \neq o$. In this case, $D(i) \neq \emptyset$ by the rules of the algorithm, and it contains the labels of some paths Π_{oi}^{ζ}, $\zeta = 1, \ldots, |D(i)|$, starting from node o and ending to node i. In addition, we have that $D(i)$ is an efficient set, by the induction hypothesis. If by using the labels belonging to $D(i)$, a new set $\bar{D}(j)$ is obtained, for some node j, such that $j \in FS(i)$, as a result of the iteration, $\bar{D}(j)$ is set to $D(j) \cup \{\bar{l}_j^1, \ldots, \bar{l}_j^m\}$, $m \leq |D(i)|$, where $\bar{l}_j^{\xi}, \xi = 1, \ldots, m$ is not dominated by the labels already belonging to $D(j)$ and the corresponding path Π_{oj}^{ξ}, consisting of Π_{oi}^{ζ} followed by the arc (i, j), is feasible. In addition, by the rules of the algorithm, the labels dominated (if they exist) by the labels $\bar{l}_j^{\xi}, \xi = 1, \ldots, m$ are deleted from $D(j)$.

  Thus, property 1b holds at the end of the iteration, completing the induction proof.

2. The proof is identical to the proof of Proposition 4.3.7.

□

The strategies used to select, at each iteration, the node to be treated are described in what follows.

DP This strategy resembles the one proposed in [115]. In particular, the node at the top of L is selected. In addition, a node i is added to the bottom of L if it is considered not labeled (i.e., at the previous iteration $D(i) = \emptyset$) otherwise it is inserted at the top.

SALF A node is inserted into the list L by following the Small Average Label First rule. Let i be the node at the top of L and let j be the node to be inserted, the operations executed by *SALF* rule are described in what follows:

```
if $\sigma_j^\xi \leq \sigma_i^\zeta$ then
    Insert $j$ at the top of $L$.
else
    Insert $j$ at the bottom of $L$.
end if
```

The node extracted is that one at the top of L.

LALL This is the Large Average Label Last strategy, described in what follows. Let i be the node at the top of L and let $s = \frac{\sum_{j \in L} \sigma_j}{|L|}$, where $\sigma_j = \frac{\sum_{l_j^\xi \in D(j)} z_j^\xi}{|D(j)|}$, *LALL* executes the operations reported below:

```
repeat
    Move $i$ to the bottom of $L$.
until $\sigma_i \leq s$
Select the node at the top of $L$.
```

SALS The selected node i is the one for which the following condition is satisfied $i = argmin_{j \in L} \{\sigma_j\}$.

$LALL - SALF$ This rule is a hybrid strategy, that has been defined combining the selection strategy $LALL$ with the insertion rule $SALF$.

Complexity Analysis

In this Section, we investigate the theoretical complexity of the proposed approaches.

Given the directed graph $\mathcal{D}(\mathcal{N}, \mathcal{A})$, the labels generated can never be more than $k|\mathcal{N}|$, where k denotes the number of efficient and feasible paths, that reach each node $j \in \mathcal{N}$ starting from node o. It is worth observing that the value of k may differ per node. In the case of $\mathcal{LSM}$, the list L can contain at most $k|\mathcal{N}|$ elements. On the other hand, a number of $|\mathcal{N}|$ elements can be stored into list L in the case of $\mathcal{NSM}$. As far as the computational complexity of the aforementioned selection strategies is concerned, we have that, for BF, DP and SLF strategies, selecting an element from L takes $\mathcal{O}(1)$. Adding a new element takes $\mathcal{O}(1)$ for BF strategy, while in the case of DP and SLF strategies a further operation is required to determine the position where the element should be added.

For the $LIFO$, LLL and SLS, the selection strategies differ in the way the element is selected from L, while adding an element takes $\mathcal{O}(1)$. The $LIFO$ strategy takes $\mathcal{O}(1)$ to add the elements, LLL takes $\mathcal{O}(\frac{|L|}{2})$ while $\mathcal{O}(|L|^2)$ operations are needed to select an element with the SLS strategy.

Let $\mathcal{O}(\beta)$ [$\mathcal{O}(\gamma)$] be the complexity for selecting [adding] an element from [to] the list L of a given selection [insertion] strategy, the following theoretical results hold.

Lemma 4.3.9. *The computational cost of the proposed $\mathcal{LSM}$ and $\mathcal{NSM}$ when applied to $\mathcal{D}(\mathcal{N}, \mathcal{A})$ is $\mathcal{O}(k^2|\mathcal{A}|\,|F| + k|\mathcal{N}|\,\beta)$, in the worst case.*

Proof. Both methods execute the same number of operations. The for-loop is performed $k\,|\mathcal{N}|$ times and requires the execution of $\frac{|\mathcal{A}|}{|\mathcal{N}|}$ iterations. At each iteration, calculating the amount of resource consumed takes $\mathcal{O}(|F|)$, while verifying label dominance takes $\mathcal{O}(k\,|F|)$, in the worst case. Finally, adding/replacing a label in the set $D(.)$ takes $\mathcal{O}(1)$, whereas the computational cost for adding an element into L is $\mathcal{O}(\gamma)$. Thus, the computational cost of the for-loop is equal to $\mathcal{O}(\frac{|\mathcal{A}|}{|\mathcal{N}|} \times max(|F|\,; k\,|F|\,; \gamma))= \mathcal{O}(\frac{|\mathcal{A}|}{|\mathcal{N}|}k\,|F|)$. As far as the computational cost of the selection strategy is concerned, we have that an element is selected at most $k\,|\mathcal{N}|$ times and the computational cost of each insertion is $\mathcal{O}(\beta)$. On the basis of the previous considerations, it is evident that the worst case computational complexity of the proposed approaches is $\mathcal{O}(k^2|\mathcal{A}|\,|F| + k|\mathcal{N}|\,\beta)$. □

As mentioned in [137], a precise expression for k is difficult to find. However, $k_{max} = \lfloor e(|\mathcal{N}| - 2)!\rfloor$ represents an upper bound on k, that is on the total number of paths between a source and destination in $\mathcal{D}(\mathcal{N}, \mathcal{A})$ ([136]).

The dynamic programming algorithms for the $\mathcal{ESPFP}$ explore the solutions space that, in this case, is represented by efficient and feasible labels. Consequently, the number of labels generated strongly influences the performances of these approaches.

It is possible to show that the number of efficient solutions is upper bounded by a value that is exponential with the number of resources taken into account. In particular, let C be the set of all feasible solutions and let r be the number of resources. When all weights $w^i, i = 1 \ldots r$ and the costs associated with each arc are independent variables, the following result holds:

Lemma 4.3.10. *The expected number of efficient solutions in the set C is upper bounded by* $(\ln C)^{r-1}$.

Proof. For the proof of this lemma, the reader is referred to [12] and [18]. □

It is evident that in our case $r = |F| + 1$, thus an upper bound on the dimension of the states space is $(\ln C)^{|F|}$.

4.4 Computational Experiments

An extensive computational phase has been carried out with the goal of assessing the performance of the proposed B&B and dynamic programming methods to address the $\mathcal{ESPFP}^{od}$ and the $\mathcal{ESPFP}$.

The approaches, described in the previous Sections, have been coded in Java and have been tested by using an Intel(R) Core(TM) i7 CPU M 620 PC, 2.67 GHz, RAM 4.00 GB, under Microsoft 7 operating system.

Considering the B&B approach, different search strategies have been implemented and the use of heuristic information in the search process has been also considered. Indeed, three different versions of each method have been implemented: a naive version (i.e., *nv*), in which no heuristic information are considered and two enhanced versions (i.e., 1^{st} and 2^{nd}) in which upper and lower bounds on the optimal path length are taken into account. In particular, in the 1^{st} enhanced version a lower bound is used to fathom unpromising nodes, whereas the 2^{nd} version uses both lower and upper bounds to prune some unpromising branches, during the searching process.

It is worth observing that in the case of the BeFS strategy, two different rules can be adopted to select the problem to be solved at each iteration, that is the LIFO (referred in the sequel to as $BeFS_{LF}$) and the FIFO (referred in the sequel to as $BeFS_{FF}$) strategy. In what follows, we shall refer to each implemented algorithm as *search strategy.version*. The notation $BFS.1^{st}$ represents the first enhanced version of the proposed B&B approach, that uses the breath first search strategy, whereas with $BeFS_{FF}.nv$, we indicate the naive B&B approach, that considers the best first search, based on the FIFO selection strategy.

Regarding the dynamic programming approaches, we have considered both node and label extension rule, to explore the state space. In addition, several node/label selection strategies have been defined to individuate the most promising node/label to be selected. In the sequel, each of the proposed approaches will be referred to as *extention rule.selection strategy.* Thus, at each iteration of the algorithm $\mathcal{LSM}.BF$ a label, chosen by following the BF selection strategy, is processed.

4.4.1 Test Problems

As far as the choice of the test problems is concerned, random networks of varying size and density (defined as the ratio between the number of arcs and the number of nodes) have been considered. In particular, the developed approaches have been tested on fully random networks ($R1 - R9$), generated by using the public domain Netgen generator ([95]). To avoid negative cost cycles, the shortest path problem on each generated instance is solved, by using the Bellman Ford algorithm. If a cycle with negative cost is detected, the cost of each arc belonging to the cycle is updated, using the pseudo random generator of the Netgen code. The Bellman Ford algorithm is executed on the modified network, until no cycle with negative cost is found.

The characteristics of the networks are given in Table 4.1, in which for each test problem the number of nodes, the number of arcs and the density value are reported. The arc costs have been chosen according to a uniform distribution from the range $[-100, 100]$. Starting from $R1-R9$, two sets of instances have been generated. They differ in the number of forbidden paths $|F|$ and in the strategy used to generate them. In particular, for the first set, three different values of $|F|$ have been considered, 250, 500 and 750.

The forbidden paths have been generated randomly on the basis of

Test	Nodes	Arcs	Density
R1	50	500	10
R2	400	1000	2.5
R3	100	2000	20
R4	200	2000	10
R5	100	5000	50
R6	2000	10000	5
R7	2500	10000	4
R8	1500	20000	13.33
R9	1000	30000	30

Table 4.1: Characteristics of the Fully Random Networks

a two-step procedure. In the first step, the number of nodes belonging to each path is determined, whereas the generation of the path is carried out in the second step. More specifically, in Step 1, for each path π_μ the number of nodes n_{π_μ} is randomly generated according to a uniform distribution from the range $[2, 10]$. In the Step 2, for each path π_μ, the origin node o is chosen randomly in the interval $[1, |N|]$; starting from o, $n_{\pi_\mu} - 1$ consecutive arcs are randomly selected.

In what follows, we use the notation $test^{|F|}$ to represent each test problem. Indeed, $R1^{250}$ refers to the fully random problem $R1$, with 250 forbidden paths.

Since the computational overhead of the B&B approaches depends on the number of the B&B nodes explored during the search process and, consequently, on the number of violated forbidden paths constraints in the relaxed problem, a second set of 48 test problems has been generated. In particular, for each random network, we have considered 8 different values of $|F|$, that is $|F| = 1, \dots, 8$. In this case, the forbidden paths have been generated in such a way all of them are included in the solution of the shortest path problem without restrictions. For each problem size ten different instances have been generated.

The choice of these two sets of test problems is motivated by the goal of investigating the behavior of the proposed solution approaches in two limit cases: 1) a few number of forbidden paths is included into the optimal solution of the relaxed problem; 2) all the forbidden paths constrains are violated.

The experimentations can be viewed as divided in two main phases. In the former, the behaviors of the $B\&B$ methods and the dynamic programming based algorithms are evaluated, by considering the first set of test problems, in which the forbidden paths are generated randomly; the related results are reported in Sections 4.4.2 and 4.4.3, respectively. In the latter, a comparison of the performances of the two different classes of solution approaches is carried out (see Section 4.4.4). In this Section, the numerical results obtained on the second set of test problems are also presented and discussed.

4.4.2 Numerical Results of Branch & Bound Methods

In this Section, the computational results obtained with the proposed B&B methods are presented and discussed. In the implemented algorithms, a label correcting method based on the SLF rule is used to solve the relaxed problems associated with the B&B nodes.

The results obtained when solving the $\mathcal{ESPFP}^{od}$ on the first set of test problems are given in Table 4.2, where for each network, the number of B&B nodes explored during the search and the execution time (in milliseconds) averaged on the related ten instances are reported. The best results obtained are highlighted in bold.

The numerical results clearly underline that the best performance is achieved with the second enhanced versions of the B&B approach. Indeed, the use of heuristic information (i.e., lower and upper bounds) allows to prune unpromising branches, resulting in improved performance. In particular, the average execution time of the 2^{nd} enhanced versions is equal to 1816.72 milliseconds and the average number of explored nodes is equal to 19.35, whereas when only a lower bound is used to fathom unpromising nodes the average computational cost increases to 1821.92 and the average number of solved candidate problems is equal to 20.83. The naive versions show the worst performance. Indeed, the average computational cost is equal to 1969.78 and the average number of explored problems is equal to 21.49. As far the comparison of the search strategies is concerned, the results underline that, for the naive versions, DFS outperforms BFS (i.e., it is observed an average percentage reduction in the computational cost and in the number of explored nodes of the 14.66% and 1.66%, respectively), whereas in the case of the enhanced versions, they show comparable performance.

Table 4.2 also underlines that the best performance are achieved by the BeFS search strategy, based on the LIFO selection strategy; the naive version, that follows the breath first search strategy, shows the worst performance. In particular, $BeFS_{LF}.2^{nd}$ is on average 1.19 times faster than $BFS.nv$ and solves a number of candidate problems that is 1.15 times less.

Test	**Results**	$BFS.nv$	$DFS.nv$	$BFS.1^{st}$	$BeFS_{FF}.1^{st}$	$DFS.1^{st}$	$BeFS_{LF}.1^{st}$	$BFS.2^{nd}$	$BeFS_{FF}.2^{nd}$	$DFS.2^{nd}$	$BeFS_{LF}.2^{nd}$
$R1^{250}$	B&B nodes	6.20	6.00	5.80	5.80	6.00	5.80	5.80	5.80	6.00	5.80
	time	4.70	6.20	4.70	3.20	6.20	4.60	4.70	6.40	4.70	**3.20**
$R1^{500}$	B&B nodes	6.40	6.30	6.30	6.30	6.30	6.30	6.30	6.30	6.30	6.30
	time	4.60	7.70	6.20	4.70	9.40	6.20	4.60	7.70	3.20	**1.50**
$R1^{750}$	B&B nodes	12.20	11.80	11.80	11.80	11.80	11.80	11.80	11.80	11.80	11.80
	time	10.90	9.40	10.90	10.90	11.00	10.90	**6.30**	11.00	12.40	10.90
$R2^{250}$	B&B nodes	11.30	11.20	10.90	10.90	11.20	10.90	10.90	10.90	11.20	10.90
	time	53.10	53.00	51.50	56.20	53.10	**49.90**	51.70	51.50	54.60	53.10
$R2^{500}$	B&B nodes	23.80	24.70	23.80	23.50	24.70	23.50	23.50	23.50	24.70	23.50
	time	129.40	131.00	129.50	131.00	127.90	123.20	117.20	115.50	117.00	**110.80**
$R2^{750}$	B&B nodes	36.90	39.00	34.50	34.50	38.80	34.50	34.50	34.50	37.50	34.50
	time	228.70	220.00	196.50	174.70	221.50	193.50	176.30	**169.90**	180.90	174.70
$R3^{250}$	B&B nodes	9.70	9.40	9.30	9.30	9.30	9.30	9.30	9.30	9.30	9.30
	time	31.60	28.80	29.60	30.10	30.00	30.10	28.30	28.00	28.00	**27.60**
$R3^{500}$	B&B nodes	21.10	21.10	20.00	20.00	21.00	19.80	20.00	20.00	21.00	19.80
	time	71.10	66.50	66.40	66.50	66.90	64.90	62.90	**62.20**	64.10	69.30
$R3^{750}$	B&B nodes	28.00	30.40	27.70	27.70	30.40	27.70	27.70	27.70	30.40	27.70
	time	112.60	121.50	113.10	114.00	121.40	112.40	**90.40**	91.40	98.90	94.00
$R4^{250}$	B&B nodes	15.70	16.00	15.30	15.30	15.80	15.20	15.10	15.10	15.70	15.10
	time	118.20	111.80	107.40	120.00	111.40	107.30	90.30	**89.60**	94.60	95.10
$R4^{500}$	B&B nodes	47.20	42.10	43.70	43.10	42.00	42.30	43.10	43.10	42.00	42.30
	time	311.20	278.20	292.00	323.30	278.80	281.30	272.10	272.20	**265.30**	266.70
$R4^{750}$	B&B nodes	49.00	49.00	49.00	49.00	49.00	49.00	49.00	49.00	49.00	49.00
	time	304.00	304.00	309.00	301.00	299.00	295.00	298.00	295.00	302.00	**294.00**
$R5^{250}$	B&B nodes	6.70	6.10	6.10	6.10	6.10	6.10	6.10	6.10	6.10	6.10
	time	41.20	34.70	35.10	35.20	39.80	37.90	34.80	32.80	**32.50**	32.70
$R5^{500}$	B&B nodes	60.00	56.60	55.40	56.00	55.80	59.40	51.50	9.10	9.10	9.10
	time	28.60	29.20	27.40	27.40	29.20	27.40	**27.40**	44.40	44.50	44.50
$R5^{750}$	B&B nodes	11.40	29.20	27.40	27.40	29.20	27.40	27.40	27.40	29.20	27.40
	time	164.90	180.40	169.90	177.90	183.10	170.10	**140.10**	141.10	149.80	143.20
$R6^{250}$	B&B nodes	11.50	11.50	11.50	11.50	11.50	11.50	11.50	11.50	11.50	11.50
	time	2311.30	**2159.20**	2184.20	2264.90	2255.90	2259.00	2213.70	2302.50	2269.90	2264.90
$R6^{500}$	B&B nodes	29.50	29.80	29.50	29.50	29.80	29.50	29.50	29.50	29.80	29.50
	time	5403.90	**5222.80**	5251.10	5403.40	5378.90	5408.60	5322.80	5438.10	5497.40	5403.40
$R6^{750}$	B&B nodes	49.30	36.10	36.70	36.70	35.70	35.70	35.50	35.50	35.50	35.50
	time	9216.50	**6236.80**	6487.90	6375.80	6303.80	6385.20	6321.20	6458.30	6397.50	6375.80
$R7^{250}$	B&B nodes	10.90	11.20	10.90	10.90	11.20	10.90	10.90	10.90	11.20	10.90
	time	2419.60	2330.60	2277.70	2338.40	2402.50	2319.80	2339.90	2380.60	2449.10	**2254.20**
$R7^{500}$	B&B nodes	26.20	25.30	25.60	25.30	25.30	25.30	25.30	25.30	25.30	25.30
	time	5717.50	5226.00	5247.80	5379.10	5322.70	5375.70	5307.30	5492.80	5447.40	**5176.10**
$R7^{750}$	B&B nodes	52.90	46.30	47.40	46.50	45.90	45.60	46.40	46.40	45.80	45.50
	time	11322.60	9292.90	9550.40	9506.90	9475.60	9492.60	9588.00	9779.70	9602.60	**9260.20**
$R8^{250}$	B&B nodes	8.60	8.30	8.30	8.30	8.30	8.30	8.30	8.30	8.30	8.30
	time	2709.80	**2533.30**	2553.70	2625.50	2606.70	2639.70	2599.00	2678.60	2672.00	2539.60
$R8^{500}$	B&B nodes	11.30	11.00	9.30	9.30	10.80	9.30	9.20	9.20	10.70	9.20
	time	3614.50	3434.90	**2865.70**	2914.10	3377.30	2970.40	2911.10	2990.70	3429.30	2868.90
$R8^{750}$	B&B nodes	20.40	22.10	19.50	19.50	22.10	19.50	19.50	19.50	22.10	19.50
	time	6475.40	6782.80	6158.80	6056.00	6905.80	6076.40	6040.40	6236.90	7061.10	**5928.20**
$R9^{250}$	B&B nodes	4.00	4.30	3.80	3.80	4.10	3.80	3.60	3.60	3.90	3.60
	time	1257.10	1305.60	1216.80	1107.60	1280.70	1178.00	1116.90	1176.20	1226.20	**1107.60**

continued on next page

continued from previous page

Test	**Results**	$BFS.nv$	$DFS.nv$	$BFS.1^{st}$	$BeFS_{FF}.1^{st}$	$DFS.1^{st}$	$BeFS_{LF}.1^{st}$	$BFS.2^{nd}$	$BeFS_{FF}.2^{nd}$	$DFS.2^{nd}$	$BeFS_{LF}.2^{nd}$
$R9^{500}$	B&B nodes	4.80	4.80	4.20	4.20	4.80	4.20	4.10	4.10	4.70	4.10
	time	1623.90	1558.40	1452.40	1368.30	1589.90	1391.70	1394.50	1387.00	1583.50	**1368.30**
$R9^{750}$	B&B nodes	10.00	6.10	7.20	6.00	5.50	5.80	5.60	5.30	5.30	5.30
	time	3129.20	1886.40	2313.40	1656.90	1750.50	1834.70	1764.40	1689.60	1738.00	**1656.90**
Average nodes		**21.67**	**21.32**	**20.77**	**20.67**	**21.20**	**20.68**	**20.42**	**18.84**	**19.39**	**18.77**
Average time		**2104.30**	**1835.26**	**1818.86**	**1799.00**	**1860.70**	**1809.13**	**1789.79**	**1830.73**	**1882.46**	**1763.90**

Table 4.2: Computational results obtained when solving $\mathcal{ESPFP}^{od}$ with B&B approaches on the first set of test problems.

Test	**Results**	$BFS.nv$	$DFS.nv$	$BFS.1^{st}$	$BeFS_{FF}.1^{st}$	$DFS.1^{st}$	$BeFS_{LF}.1^{st}$	$BFS.2^{nd}$	$BeFS_{FF}.2^{nd}$	$DFS.2^{nd}$	$BeFS_{LF}.2^{nd}$
$R1^{250}$	B&B nodes	25.00	24.20	24.10	25.10	24.20	24.20	13.50	13.50	13.50	13.50
	time	31.10	37.40	37.20	34.30	37.50	34.20	15.60	**14.10**	16.60	18.60
$R1^{500}$	B&B nodes	33.00	32.90	32.70	43.00	32.90	32.90	21.80	21.80	22.00	22.00
	time	51.50	46.70	55.50	48.40	48.30	51.50	24.90	28.20	**24.40**	26.40
$R1^{750}$	B&B nodes	57.30	66.70	56.90	64.40	66.70	64.70	12.70	12.70	12.70	12.70
	time	84.20	68.70	78.60	76.60	70.10	76.40	20.30	20.10	18.20	**18.00**
$R2^{250}$	B&B nodes	45.50	45.50	45.50	45.50	45.50	45.50	45.50	45.50	45.50	45.50
	time	617.70	622.90	597.70	605.30	600.70	586.70	480.60	502.30	**454.30**	458.20
$R2^{500}$	B&B nodes	125.50	125.20	125.50	125.50	125.20	125.20	125.20	125.20	125.20	125.20
	time	2037.50	2012.30	2072.40	2076.30	2109.10	1903.40	1829.90	1812.70	**1693.20**	1754.60
$R2^{750}$	B&B nodes	327.00	328.20	326.70	327.00	328.20	330.90	326.70	327.00	328.20	330.90
	time	6829.80	7095.10	6962.10	6988.80	7263.50	6464.80	6829.80	6766.00	**6180.30**	6808.00
$R3^{250}$	B&B nodes	11.20	11.20	11.20	11.20	11.20	11.20	11.20	11.20	11.20	11.20
	time	70.50	68.50	74.20	72.80	69.20	70.80	48.10	47.90	48.30	**46.30**
$R3^{500}$	B&B nodes	29.90	28.00	29.90	28.70	28.00	28.00	29.90	28.70	28.00	28.00
	time	214.70	198.30	214.20	204.90	200.20	199.60	171.50	162.70	158.20	**155.00**
$R3^{750}$	B&B nodes	267.60	218.30	267.60	193.80	218.30	218.30	240.60	166.80	190.10	190.10
	time	1651.70	1380.30	1568.60	1278.90	1382.40	1397.00	1108.90	**796.90**	952.70	889.10
$R4^{250}$	B&B nodes	205.14	98.86	205.14	226.29	98.86	107.71	185.43	206.57	177.25	81.14
	time	7918.71	2697.71	7916.00	8381.57	2573.71	3006.57	7677.71	8234.00	3194.50	**2563.57**
$R4^{500}$	B&B nodes	765.33	550.56	644.44	575.11	550.56	550.56	607.11	523.11	436.33	436.33
	time	29252.11	14623.33	15660.89	13503.22	14300.00	14399.56	14933.33	12223.11	**10767.00**	10817.00
$R4^{750}$	B&B nodes	1712.00	912.00	1712.00	1691.00	912.00	952.00	1239.00	307.00	912.00	952.00
	time	13159.00	5931.00	13067.00	12834.00	5925.00	5952.00	6145.00	**1420.00**	5004.00	5283.00
$R5^{250}$	B&B nodes	10.00	10.00	10.00	10.00	10.00	10.00	7.20	7.20	7.20	7.20
	time	74.80	72.40	71.80	71.90	71.80	72.10	41.00	**37.70**	38.70	37.90
$R5^{500}$	B&B nodes	16.10	15.80	15.90	15.90	15.80	15.80	15.80	15.80	15.80	15.80
	time	156.20	154.30	149.70	149.70	155.30	148.40	108.70	**105.90**	113.20	109.00
$R5^{750}$	B&B nodes	265.50	96.30	91.80	67.20	96.30	96.30	91.80	67.20	96.30	96.30
	time	4159.80	967.50	999.30	642.50	964.80	965.70	887.00	**592.20**	938.60	912.20
$R6^{250}$	B&B nodes	53.40	53.10	53.10	53.40	53.10	53.10	53.10	53.40	53.10	53.10
	time	9644.00	9634.60	9717.90	9754.70	9790.70	9174.30	9122.70	9007.50	**8456.90**	8700.80
$R6^{500}$	B&B nodes	228.00	228.00	228.00	228.00	228.00	228.00	228.00	228.00	228.00	228.00
	time	45892.20	42748.70	45449.00	46501.90	44071.80	41917.20	46280.50	45505.40	**39946.60**	41185.30
$R6^{750}$	B&B nodes	729.30	723.60	727.50	703.50	723.60	723.60	727.50	703.50	723.60	723.60
	time	262090.10	260438.40	252863.70	262251.90	273453.20	244444.70	262428.30	258647.40	**235439.30**	261816.50
$R7^{250}$	B&B nodes	58.80	58.80	58.80	58.80	58.80	58.80	58.80	58.80	58.80	58.80
	time	14673.40	12780.10	14002.80	14534.70	13570.50	13267.90	13621.70	13249.00	**11829.10**	12049.80
$R7^{500}$	B&B nodes	214.40	214.40	214.40	214.40	214.40	214.40	214.40	214.40	214.40	214.40
	time	54116.70	47754.30	51842.00	53874.50	50508.40	48403.80	53176.00	50768.40	**44503.90**	47494.60
$R7^{750}$	B&B nodes	582.00	591.90	582.00	598.20	591.70	591.60	582.00	598.20	582.00	582.00
	time	298611.30	273213.30	292979.30	293511.30	284193.40	262815.10	293227.60	286659.80	**260004.10**	281291.80
$R8^{250}$	B&B nodes	47.60	51.50	47.60	47.60	51.50	51.50	47.60	47.60	52.40	52.40
	time	17990.10	15921.40	21933.80	17891.80	16408.00	15707.80	17397.30	17088.30	**15065.00**	15694.40
$R8^{500}$	B&B nodes	102.70	102.70	102.70	104.60	102.70	102.70	88.90	90.70	88.90	88.90
	time	47843.70	41728.70	56546.80	45461.60	45171.40	42311.80	27711.90	27126.90	**23123.80**	23134.90
$R8^{750}$	B&B nodes	249.10	226.50	245.20	247.30	226.50	227.70	218.20	219.20	209.80	209.80
	time	78821.50	72143.90	84862.50	77948.90	74139.30	69787.00	76162.50	74836.60	**63354.60**	66756.00
$R9^{250}$	B&B nodes	49.30	49.30	49.30	49.30	49.30	49.30	49.30	49.30	49.30	49.30
	time	10036.90	10118.30	10416.90	10461.40	10631.40	9966.90	10165.00	10063.40	**9617.40**	9933.60

continued on next page

continued from previous page

Test	**Results**	$BFS.nv$	$DFS.nv$	$BFS.1^{st}$	$BeFS_{FF}.1^{st}$	$DFS.1^{st}$	$BeFS_{LF}.1^{st}$	$BFS.2^{nd}$	$BeFS_{FF}.2^{nd}$	$DFS.2^{nd}$	$BeFS_{LF}.2^{nd}$
$R9^{500}$	B&B nodes	116.40	91.20	112.50	106.00	91.20	92.10	86.40	87.60	69.90	71.10
	time	20339.10	25814.60	21622.20	21054.30	26036.50	21919.80	15829.60	**15322.40**	17086.70	18096.90
$R9^{750}$	B&B nodes	111.10	104.20	105.20	110.90	104.00	104.00	103.60	103.60	102.10	102.00
	time	29498.00	28080.10	31667.80	30398.40	30794.50	29554.40	31184.40	29861.50	**27994.80**	28827.70
Average nodes		**238.45**	**187.37**	**226.88**	**221.17**	**187.35**	**189.26**	**201.16**	**160.50**	**179.76**	**177.83**
Average time		**35402.46**	**32457.51**	**34941.85**	**34467.21**	**33871.88**	**31281.46**	**33208.51**	**32255.57**	**29112.01**	**31291.82**

Table 4.3: Computational results obtained when solving $\mathcal{ESPFP}$ with $B\&B$ approaches on the first set of test problems.

Table 4.3 provides statistics on the solution process of the proposed $B\&B$ approaches, when solving $\mathcal{ESPFP}$ on the first set of test problems. Similarly to that done for the $\mathcal{ESPFP}^{od}$, for each test problem and for each method, we report the average computational cost and the average number of $B\&B$ nodes explored during the search process.

The results collected underline that the LIFO search strategy outperforms the FIFO counterpart for both the enhanced and the naive versions. In particular, $DFS.nv$, $DFS.1^{st}$ and $DFS.2^{nd}$ are 1.09, 1.03 and 1.14 times faster than $BFS.nv$, $BFS.1^{st}$ and $BFS.2^{nd}$, respectively. This behaviour can be justified by considering the number of $B\&B$ nodes explored, that for $DFS.nv$, $DFS.1^{st}$ and $DFS.2^{nd}$ is 1.27, 1.21 and 1.11 times greater than that obtained by $BFS.nv$, $BFS.1^{st}$ and $BFS.2^{nd}$, respectively.

Referring to the enhanced versions, the computational results demonstrate that for both the versions, $BeFS_{LF}$ [$BeFS_{FF}$] and DFS [BFS] show comparable performances.

From Table 4.3, it is evident that also in this case, the use of heuristic information allows to cut-off unpromising candidate problems. In general, the enhanced versions outperform the naive counterparts. Indeed, the average execution time of the best performing 2^{nd} enhanced version (i.e., $DFS.2^{nd}$) is 29112.01 milliseconds, whereas the average computational cost of the naive versions is about 22929.98 milliseconds. This behaviour can be explained by comparing the number of problems solved by the two versions of each algorithm: the average percentage reduction in the number of nodes explored by $DFS.2^{nd}$ with the respect to the naive versions is of 34.09%.

4.4.3 Numerical Results of Dynamic Programming Approaches

The computational experiments have been carried out with the aim of assessing the performance of the proposed dynamic programming approaches and individuating the most efficient version, when solving the considered test problems. The related numerical results are reported in Tables 4.4 and 4.5, in which for each network and for each algorithm the number of iterations and the execution time (in milliseconds) averaged on the corresponding ten instances are given. Also in this case, the best results in terms of computational effort are highlihted in bold.

As shown in Table 4.4, the best selection strategy, based on label extension, is BF. The average computational cost of $\mathcal{LSM}.BF$ is equal to 1657.71 milliseconds with 4263.48 iterations, on average. $\mathcal{LSM}.DP$ shows the worst performances and it is 14.20 times slower than $\mathcal{LSM}.BF$; the number of iteration executed by $\mathcal{LSM}.BF$ is 13.39 times less that that performed by $\mathcal{LSM}.DP$.

Considering the node selection methods, $\mathcal{NSM}.LALL$ behaves the best, while the worst performances are shown by $\mathcal{NSM}.LIFO$. In particular, $\mathcal{NSM}.LIFO$ is, on average, 37.35 times slower than $\mathcal{NSM}.LALL$. This behaviour can be justified considering the number of iterations executed by $\mathcal{NSM}.LALL$ that is 23.55 times less than that performed by $\mathcal{NSM}.LIFO$.

Comparing the best selection strategy of each extension rule, $\mathcal{LSM}.BF$ and $\mathcal{NSM}.LALL$ show similar performance. Indeed, the former is 1.04 times faster than the latter.

4.4.4 Comparison

The goal of this Section is to provide a comparison of the performances of the two classes of solution approaches defined to solve the problem under study.

It is worth observing that the dynamic programming approaches solve both the single-origin single-destination (i.e., $\mathcal{ESPFP}^{od}$) and the single-origin all-destination (i.e., $\mathcal{ESPFP}$) versions of the problem, within the same computa-

Test	**Results**	$\mathcal{LSM}.BF$	$\mathcal{LSM}.DP$	$\mathcal{LSM}.SLF$	$\mathcal{LSM}.LIFO$	$\mathcal{LSM}.LLL$	$\mathcal{LSM}.SLS$	$\mathcal{LSM}.LLL\text{-}SLF$
$R1^{250}$	iter	153.50	635.80	152.30	613.50	138.80	133.50	139.30
	time	6.30	65.60	8.50	58.90	7.80	**6.00**	6.70
$R1^{500}$	iter	183.00	712.20	181.00	687.90	162.60	157.20	164.80
	time	**9.40**	138.90	10.60	127.30	9.70	11.30	10.40
$R1^{750}$	iter	189.40	558.10	188.10	531.90	169.80	163.10	172.80
	time	20.30	75.00	19.10	70.50	**16.50**	17.80	18.40
$R2^{250}$	iter	4864.00	12654.40	4772.60	12429.20	4736.60	4726.10	4732.90
	time	857.30	3302.40	844.70	3223.00	**834.30**	851.40	850.40
$R2^{500}$	iter	8526.40	18627.10	8404.20	18371.70	8358.80	8333.30	8341.70
	time	**4601.90**	15400.30	4694.50	15304.40	4669.10	4673.40	4624.60
$R2^{750}$	iter	11627.90	26617.30	11478.90	26361.80	11410.10	11394.70	11403.10
	time	11160.50	44257.10	10695.90	43600.80	10701.20	**10413.40**	10470.70
$R3^{250}$	iter	1294.60	7145.00	1186.20	7105.50	1039.20	847.70	1011.60
	time	519.00	930.75	614.00	885.50	475.90	**443.70**	539.10
$R3^{500}$	iter	718.30	1333.23	664.20	1325.86	616.90	553.20	609.70
	time	95.00	173.67	99.60	165.23	**89.00**	106.70	93.30
$R3^{750}$	iter	1900.29	12337.42	1801.29	12269.22	1739.00	1641.29	1718.57
	time	778.14	1607.15	**756.86**	1529.01	806.43	1349.86	783.86
$R4^{250}$	iter	1405.30	1005.16	1186.30	999.60	1113.00	875.90	1038.50
	time	79.60	130.94	71.10	124.57	70.20	77.50	**66.20**
$R4^{500}$	iter	11077.50	148254.89	10501.50	147435.29	9902.38	8693.63	9598.13
	time	**6859.38**	19312.56	8662.75	18373.65	10594.38	15871.25	11788.63
$R4^{750}$	iter	13334.67	216446.15	13379.67	215249.56	12654.00	11535.33	12316.33
	time	**9603.67**	28195.56	13303.17	26824.78	15576.67	24571.17	15456.67
$R5^{250}$	iter	373.10	6024.00	343.90	5961.13	304.90	265.20	302.50
	time	62.40	2212.63	59.60	2152.88	**53.60**	56.50	55.00
$R5^{500}$	iter	420.60	9776.10	387.20	9715.60	344.30	299.40	340.90
	time	108.90	8931.30	107.10	9047.60	**91.00**	96.40	95.00
$R5^{750}$	iter	1780.40	6781.29	1673.50	6718.71	1574.90	1377.90	1546.10
	time	**1815.70**	3812.14	1859.80	3639.86	1864.50	3387.40	2021.40
$R6^{250}$	iter	3974.90	22954.80	3561.10	22780.50	3239.80	3074.40	3187.00
	time	293.40	1748.70	222.60	1326.60	206.80	**201.30**	204.10
$R6^{500}$	iter	4334.90	59331.11	3876.00	59117.44	3584.30	3366.60	3492.30
	time	393.20	15311.33	294.50	14955.00	296.00	290.80	**276.20**
$R6^{750}$	iter	5858.60	175538.90	5327.90	175295.60	4823.90	4594.00	4777.60
	time	638.00	175833.10	533.60	176574.10	**519.30**	540.30	536.90
$R7^{250}$	iter	5348.60	70091.00	4849.10	69834.50	4452.80	4309.80	4412.10
	time	324.60	6307.50	279.00	5221.30	289.10	350.00	**270.20**
$R7^{500}$	iter	7754.40	148732.20	7108.40	148402.00	6471.20	6249.10	6405.10
	time	606.90	35616.90	557.10	34151.30	**542.00**	560.30	552.70
$R7^{750}$	iter	9442.30	389962.60	8661.30	389609.50	7998.60	7768.50	7941.10
	time	915.90	199805.20	844.00	203575.90	**811.70**	817.30	854.90
$R8^{250}$	iter	3617.50	27198.90	3261.60	27055.40	2964.70	2503.70	2844.60
	time	527.10	3746.60	482.10	3379.20	446.70	**369.60**	429.60
$R8^{500}$	iter	3791.20	34616.50	3482.20	34476.90	3095.50	2629.30	3009.40
	time	694.00	6123.80	641.30	5849.70	569.70	**490.40**	568.00
$R8^{750}$	iter	4197.30	69766.90	3813.70	69630.50	3395.70	2855.50	3311.60
	time	906.30	34172.50	851.50	34272.90	735.90	**678.40**	765.40
$R9^{250}$	iter	2717.80	19709.60	2581.70	19592.70	2324.50	1800.80	2270.40
	time	655.30	4626.40	657.50	4386.70	562.20	**459.30**	573.00
$R9^{500}$	iter	2867.30	26087.00	2609.50	25970.90	2439.10	1848.20	2319.20
	time	900.10	8230.20	850.50	7839.10	765.10	**617.90**	756.10
$R9^{750}$	iter	3360.10	28309.10	3234.80	28180.00	2908.50	2364.20	2863.90
	time	1325.90	15456.50	1341.90	14406.60	1139.20	**1045.40**	1174.60
Average iter		**4263.48**	**57081.73**	**4024.75**	**56878.61**	**3776.44**	**3494.87**	**3713.75**
Average time		**1657.71**	**23537.95**	**1828.25**	**23372.83**	**1953.48**	**2531.66**	**1994.15**

Table 4.4: Computational results obtained when solving $\mathcal{ESPFP}$ with $\mathcal{LSM}$ on the first set of test problems.

tional time.

Test	**Results**	$\mathcal{NSM}.BF$	$\mathcal{NSM}.DP$	$\mathcal{NSM}.SALF$	$\mathcal{NSM}.LIFO$	$\mathcal{NSM}.LALL$	$\mathcal{NSM}.SALS$	$\mathcal{NSM}.LALL\text{-}SALF$
$R1^{250}$	iter	82.10	154.40	92.50	400.10	81.10	82.10	83.50
	time	8.70	12.10	7.80	60.30	8.20	**5.50**	7.60
$R1^{500}$	iter	86.30	172.70	93.60	382.20	86.70	89.20	87.00
	time	11.00	18.00	12.90	75.80	**10.40**	11.10	11.60
$R1^{750}$	iter	90.00	170.80	98.50	404.60	86.40	90.20	90.00
	time	23.60	27.90	23.20	79.10	**18.20**	19.20	20.20
$R2^{250}$	iter	1830.20	3392.40	1342.40	11158.60	1376.70	1284.20	1319.10
	time	938.20	1534.20	818.00	4304.20	825.80	816.00	**810.90**
$R2^{500}$	iter	2363.10	4525.90	1655.00	16033.30	1684.90	1555.00	1593.50
	time	4816.30	7521.40	4638.10	20110.50	4782.10	4680.90	**4578.70**
$R2^{750}$	iter	2782.70	6409.40	1860.80	22568.90	1899.00	1780.10	1775.10
	time	10946.00	17004.40	10505.90	57565.50	10374.50	10358.50	**10268.30**
$R3^{250}$	iter	268.50	9056.70	284.50	3414.67	254.10	247.10	252.30
	time	505.50	65690.50	**432.10**	884.83	515.20	620.70	649.30
$R3^{500}$	iter	295.80	640.90	313.10	2185.00	287.90	273.00	276.90
	time	121.80	294.20	125.00	473.00	**108.00**	109.70	109.50
$R3^{750}$	iter	373.14	1159.86	404.86	473.94	419.29	392.14	410.00
	time	1012.14	2919.86	1090.14	2189.32	937.43	**916.14**	975.00
$R4^{250}$	iter	612.10	2064.80	606.50	46.03	530.90	517.40	519.80
	time	90.90	366.70	93.70	212.64	71.20	**69.10**	70.90
$R4^{500}$	iter	990.38	13014.88	1063.75	7435.28	1082.71	1250.00	1161.50
	time	6835.63	91327.13	8926.13	34346.92	**2709.86**	6848.75	6517.38
$R4^{750}$	iter	1109.67	15150.33	1218.67	11985.93	1345.83	1569.67	1364.17
	time	**11550.83**	138709.17	11958.67	55368.42	11628.67	12960.17	11738.50
$R5^{250}$	iter	258.20	430.80	254.80	2468.22	219.80	216.60	223.50
	time	70.10	159.90	79.70	1837.78	62.60	**62.30**	65.80
$R5^{500}$	iter	266.50	447.40	262.10	2943.30	223.30	213.50	235.10
	time	131.20	219.20	147.40	5555.60	107.50	**106.00**	110.30
$R5^{750}$	iter	327.00	1104.60	346.10	5604.17	347.10	365.70	401.60
	time	**2110.50**	6835.00	3017.90	707637.33	2288.60	2544.20	2311.60
$R6^{250}$	iter	3306.70	9957.30	3221.70	22686.50	2795.30	2636.50	2761.70
	time	370.80	1574.00	358.20	1238.20	298.50	**280.40**	296.40
$R6^{500}$	iter	3370.60	11214.10	3469.30	56848.56	2872.60	2730.80	2919.00
	time	487.00	2207.30	471.60	20586.11	394.30	**362.70**	393.70
$R6^{750}$	iter	4108.10	13553.40	4082.10	166591.70	3418.30	3260.10	3440.70
	time	810.30	3091.60	773.60	252163.00	639.30	**595.00**	643.90
$R7^{250}$	iter	4403.70	14508.70	4299.90	69168.00	3717.50	3582.00	3720.00
	time	492.70	2825.50	447.60	5544.20	403.70	**381.10**	395.00
$R7^{500}$	iter	5572.70	28992.00	5458.50	144954.90	4735.90	4623.00	4777.50
	time	915.90	8810.10	838.00	46162.50	753.20	**712.20**	713.60
$R7^{750}$	iter	6311.80	74453.60	6156.50	370544.30	5641.50	5519.70	5609.30
	time	1326.20	113664.00	1247.70	291413.10	1124.50	**1092.40**	1096.20
$R8^{250}$	iter	2827.70	7640.40	2787.40	26696.90	2500.00	2279.90	2466.50
	time	653.80	1641.80	667.20	3058.90	574.30	**521.20**	593.80
$R8^{500}$	iter	2849.50	8377.20	2839.50	33454.90	2545.10	2313.70	2522.10
	time	819.70	2132.00	821.00	6369.00	730.50	**690.10**	716.20
$R8^{750}$	iter	2983.20	9263.60	2959.50	63764.70	2637.80	2385.00	2602.50
	time	1022.20	2700.70	1026.50	46201.20	887.00	**838.90**	888.20
$R9^{250}$	iter	2186.90	5269.60	2189.10	19147.80	2088.40	1885.40	2100.70
	time	698.60	1541.20	705.70	3951.90	646.10	**584.70**	621.40
$R9^{500}$	iter	2227.80	5782.30	2233.10	24410.60	2083.10	1926.50	2064.50
	time	941.40	2141.70	958.50	8130.00	842.30	**818.50**	864.30
$R9^{750}$	iter	2379.90	5893.40	2335.70	26210.30	2253.10	2133.30	2264.70
	time	1386.70	2827.00	1331.00	18853.90	1285.60	**1261.10**	1312.10
Average iter		**2009.79**	**9363.02**	**1923.31**	**41184.57**	**1748.68**	**1674.14**	**1742.31**
Average time		**1818.43**	**17696.17**	**1908.27**	**59050.86**	**1593.61**	**1787.65**	**1732.61**

Table 4.5: Computational results obtained when solving $\mathcal{ESPFP}$ with $\mathcal{NSM}$ on the first set of test problems.

First of all we focus on $\mathcal{ESPFP}^{od}$. The results collected underline that the best version of the $B\&B$ approach, that is $BeFS_{LF}.2^{nd}$ shows similar performance to that achieved by both the best version of $\mathcal{LSM}$ and $\mathcal{NSM}$ (i.e., $\mathcal{LSM}.BF$ and $\mathcal{NSM}.LALL$). In particular, $BeFS_{LF}.2^{nd}$ is 1.06 and 1.10 times slower than $\mathcal{LSM}.BF$ and $\mathcal{NSM}.LALL$, respectively.

It is worth observing that for the test problems $R1 - R5$, the B&B approach behaves always the best. Indeed, the dynamic programming based solution approaches are 22.84 times slower than the B&B solution methods (see Tables 4.2 - 4.5).

Regarding the $\mathcal{ESPFP}$, both the best versions of $\mathcal{LSM}$ and $\mathcal{NSM}$ outperform the best version of the $B\&B$ approach. In particular, $DFS.2^{nd}$ is 17.86 and 18.26 times slower than $\mathcal{LSM}.BF$ and $\mathcal{NSM}.LALL$, respectively. However, it is important to point out that 7 out of 15 instances related to the test problems $R1-R5$ are solved by the B&B solution approaches faster than the dynamic programming approaches. Indeed, on the instances related to the test problems $R1 - R5$ the latter are 1.26 times slower than the former.

The computational results on the second set of instances are reported in Table 4.7.

The collected data emphasize the superiority, in terms of computational efficiency, of the dynamic programming approaches on the second set of test problems. The best performing version of both $\mathcal{LSM}$ (i.e.: $\mathcal{LSM}.BF$) and $\mathcal{NSM}$ (i.e.: $\mathcal{NSM}.LALL$) outperform the best version of the $B\&B$ solution approach for both the single-source single-destination and single-source all-destination versions of the problem. In particular, $\mathcal{LSM}.BF$ [$\mathcal{NSM}.LALL$] is 238.09 [160.54] and 6465.73 [4359.85] time faster than $BeFSLF.2^{nd}$ and $DFS.2^{nd}$, when solving the $\mathcal{ESPFP}^{od}$ and the single-source all-destination version, respectively. In addition, the higher the number of violated constraints related to the forbidden path, the higher the speed up. Indeed, when $\mathcal{ESPFP}^{od}$ [$\mathcal{ESPFP}$] is solved, $\mathcal{LSM}.BF$ is 5.86, 11.53, 23.73, 44.83,

$\|F\|$	$\mathcal{ESPFP}^{od}$	$\mathcal{ESPFP}$
2	181.71 %	249.56 %
3	507.67 %	986.14 %
4	1285.84 %	3091.45 %
5	3107.37 %	9242.48 %
6	6871.09 %	25404.13 %
7	12781.45 %	72393.27 %
8	27960.47 %	147537.59 %

Table 4.6: Percentage increase of the explored *B&B* nodes with respect to the instances with only one violated constrain.

101.92, 191.95, 378.22 and 754.72 [4.56, 14.24, 46.02, 146.96, 495.12, 1881.61, 11115.59 and 25552.46] times faster than the *B&B* solution strategy when 1, 2, 3, 4, 5, 6, 7 and 8 forbidden paths are considered, respectively. The same behavior is observed for $\mathcal{NSM}.LALL$.

The bad performances of the *B&B* method can be justified by observing that the higher the number of violated constrains, the dramatically higher the explored *B&B* nodes. In Table 4.6, the percentage increase of the explored *B&B* nodes with respect to the instances with only one violated constrain are reported.

test	*results*	$\mathcal{LSM}.BF$	$\mathcal{NSM}.LALL$	*B&B best*	
				$\mathcal{ESPFP}^{od}$	$\mathcal{ESPFP}$
$R1^1$	iter / B&B nodes	66.20	67.90	2.80	2.80
	time	1.80	3.30	**1.70**	2.10
$R1^2$	iter / B&B nodes	67.60	68.30	5.50	5.70
	time	**0.80**	1.50	3.50	2.90
$R1^3$	iter / B&B nodes	68.20	68.10	8.20	8.10
	time	**0.80**	1.00	4.70	4.70
$R1^4$	iter / B&B nodes	69.00	68.20	8.50	9.30
	time	**0.70**	0.90	4.90	5.00
$R1^5$	iter / B&B nodes	69.00	68.10	10.00	14.10
	time	**0.80**	1.10	4.40	9.70
$R1^6$	iter / B&B nodes	69.20	68.80	13.00	28.90
	time	**0.60**	1.00	5.70	18.50
$R1^7$	iter / B&B nodes	73.10	70.50	24.70	83.20
	time	**0.70**	1.00	10.50	48.70
$R1^8$	iter / B&B nodes	72.20	69.40	24.70	98.80
	time	**1.00**	1.00	10.70	81.50
$R2^1$	iter / B&B nodes	437.80	429.60	4.00	4.00
	time	**4.00**	6.20	17.80	11.30
$R2^2$	iter / B&B nodes	513.40	472.30	11.90	15.20
	time	**5.10**	7.00	49.90	39.10
$R2^3$	iter / B&B nodes	644.00	553.90	37.30	54.80
	time	**6.00**	8.40	142.70	145.00
$R2^4$	iter / B&B nodes	740.10	602.30	91.30	176.00
	time	**7.20**	10.70	311.80	596.60
$R2^5$	iter / B&B nodes	809.80	641.80	230.80	564.20

continued on next page

continued from previous page

test	*results*	$\mathcal{LSM}.\mathcal{BF}$	$\mathcal{NSM}.\mathcal{LALL}$	*B&B best*	
				$\mathcal{ESPFP}^{od}$	$\mathcal{ESPFP}$
	time	**9.00**	13.10	759.00	2308.70
$R2^6$	iter / B&B nodes	871.40	674.10	528.70	1364.20
	time	**8.80**	12.80	1756.00	7522.10
$R2^7$	iter / B&B nodes	976.60	728.30	1001.00	3523.00
	time	**10.90**	14.00	3596.50	34571.80
$R2^8$	iter / B&B nodes	1017.30	746.20	2308.60	9880.10
	time	**11.40**	14.70	9909.30	346104.50
$R3^1$	iter / B&B nodes	217.00	161.00	4.00	4.00
	time	7.00	**8.00**	13.00	16.00
$R3^2$	iter / B&B nodes	296.00	187.00	13.00	13.00
	time	8.00	**8.00**	43.00	40.00
$R3^3$	iter / B&B nodes	298.00	183.00	31.00	47.00
	time	10.00	**7.00**	93.00	138.00
$R3^4$	iter / B&B nodes	386.00	208.00	94.00	146.00
	time	**9.00**	10.00	279.00	284.00
$R3^5$	iter / B&B nodes	370.00	199.00	148.00	358.00
	time	10.00	**8.00**	424.00	794.00
$R3^6$	iter / B&B nodes	454.00	222.00	445.00	1023.00
	time	12.00	**11.00**	1299.00	2914.00
$R3^7$	iter / B&B nodes	454.00	224.00	445.00	1177.00
	time	11.00	**11.00**	1309.00	3046.00
$R3^8$	iter / B&B nodes	466.00	228.00	1147.00	2251.00
	time	14.00	**12.00**	3002.00	7208.00
$R4^1$	iter / B&B nodes	434.00	306.00	4.00	4.00
	time	**9.00**	12.00	22.00	25.00
$R4^2$	iter / B&B nodes	759.00	373.00	13.00	16.00
	time	**17.00**	21.00	69.00	129.00
$R4^3$	iter / B&B nodes	718.00	362.00	22.00	40.00
	time	**17.00**	19.00	118.00	425.00
$R4^4$	iter / B&B nodes	712.00	362.00	31.00	88.00
	time	**16.00**	19.00	167.00	911.00
$R4^5$	iter / B&B nodes	1016.00	427.00	94.00	250.00
	time	**26.00**	28.00	505.00	2074.00
$R4^6$	iter / B&B nodes	1096.00	454.00	166.00	718.00
	time	29.00	**28.00**	899.00	6310.00
$R4^7$	iter / B&B nodes	1098.00	426.00	277.00	1876.00
	time	30.00	**28.00**	1498.00	21739.00
$R4^8$	iter / B&B nodes	1065.00	431.00	337.00	3956.00
	time	28.00	**27.00**	1832.00	60310.00
$R5^1$	iter / B&B nodes	179.00	150.00	4.00	4.00
	time	9.00	**8.00**	12.00	12.00
$R5^2$	iter / B&B nodes	208.00	156.00	13.00	16.00
	time	12.00	**10.00**	53.00	75.00
$R5^3$	iter / B&B nodes	206.00	157.00	13.00	40.00
	time	**10.00**	11.00	54.00	154.00
$R5^4$	iter / B&B nodes	231.00	163.00	33.00	40.00
	time	**11.00**	12.00	139.00	151.00
$R5^5$	iter / B&B nodes	227.00	166.00	83.00	72.00
	time	**11.00**	11.00	346.00	258.00
$R5^6$	iter / B&B nodes	223.00	162.00	83.00	84.00
	time	12.00	**11.00**	351.00	351.00
$R5^7$	iter / B&B nodes	270.00	187.00	83.00	84.00
	time	16.00	**14.00**	348.00	359.00
$R5^8$	iter / B&B nodes	268.00	189.00	83.00	84.00
	time	**16.00**	13.00	347.00	347.00
$R6^1$	iter / B&B nodes	2493.40	2391.40	4.00	4.00
	time	123.60	213.50	**743.20**	734.10
$R6^2$	iter / B&B nodes	3013.60	2639.90	10.90	14.20
	time	**155.00**	247.00	1969.00	2347.00
$R6^3$	iter / B&B nodes	3133.30	2681.60	32.50	59.50
	time	**156.70**	253.80	5867.20	10174.00
$R6^4$	iter / B&B nodes	3782.80	3018.20	94.90	234.10
	time	**193.60**	302.20	16448.60	43624.10
$R6^5$	iter / B&B nodes	4027.10	3127.30	218.20	778.60
	time	**210.40**	330.20	37306.80	184269.20
$R6^6$	iter / B&B nodes	5030.10	3691.00	550.30	2520.00
	time	**274.10**	425.50	91033.50	1072904.30
$R6^7$	iter / B&B nodes	5442.00	3880.80	1238.80	9056.80
	time	**306.10**	460.30	202573.50	8316057.80
$R6^8$	iter / B&B nodes	6232.90	4292.50	2790.40	15125.00
	time	**355.60**	535.90	447846.10	13730303.75
$R7^1$	iter / B&B nodes	2950.00	2877.30	4.00	4.00

continued on next page

continued from previous page

test	*results*	$\mathcal{LSM}.BF$	$\mathcal{NSM}.LALL$	*B&B best*	
				$\mathcal{ESPFP}^{od}$	$\mathcal{ESPFP}$
	time	138.30	228.30	**949.90**	710.90
$R7^2$	iter / B&B nodes	3457.90	3198.80	11.80	15.50
	time	**167.20**	280.30	2690.80	3739.20
$R7^3$	iter / B&B nodes	4367.70	3702.50	26.70	55.20
	time	**218.40**	352.00	5984.20	14774.10
$R7^4$	iter / B&B nodes	5056.50	4081.10	56.40	177.80
	time	**250.70**	421.70	12374.00	53257.00
$R7^5$	iter / B&B nodes	5256.10	4167.10	131.80	576.40
	time	**273.50**	444.00	28301.50	227924.50
$R7^6$	iter / B&B nodes	5539.60	4326.50	294.70	1584.90
	time	**288.60**	460.90	65518.80	857666.90
$R7^7$	iter / B&B nodes	6336.40	4790.70	711.40	5737.60
	time	**335.60**	539.70	152955.00	6108179.90
$R7^8$	iter / B&B nodes	7114.30	5162.70	1425.10	11895.14
	time	**399.00**	618.80	297430.00	20885448.29
$R8^1$	iter / B&B nodes	2054.00	2114.50	3.40	3.40
	time	206.90	384.70	**1282.60**	855.70
$R8^2$	iter / B&B nodes	2479.60	2308.60	7.30	9.20
	time	**265.70**	437.10	2659.00	2581.90
$R8^3$	iter / B&B nodes	2600.40	2413.10	13.30	24.40
	time	**274.00**	459.40	4824.50	8831.10
$R8^4$	iter / B&B nodes	2719.60	2458.90	20.50	77.00
	time	**289.90**	472.70	7314.60	23928.60
$R8^5$	iter / B&B nodes	2731.80	2459.20	48.10	176.50
	time	**297.60**	464.50	17076.80	62527.90
$R8^6$	iter / B&B nodes	2866.90	2510.60	81.70	455.00
	time	**321.00**	480.30	29140.20	221263.00
$R8^7$	iter / B&B nodes	2924.40	2528.50	124.30	987.00
	time	**329.60**	495.50	44003.80	599944.40
$R8^8$	iter / B&B nodes	2976.40	2541.90	255.10	2822.60
	time	**337.60**	498.30	80733.40	3042016.10
$R9^1$	iter / B&B nodes	1350.20	1578.30	3.70	3.70
	time	212.90	336.10	**1132.50**	879.90
$R9^2$	iter / B&B nodes	1628.50	1745.20	9.10	13.70
	time	**259.40**	382.60	2725.10	3726.30
$R9^3$	iter / B&B nodes	1780.00	1811.40	22.00	39.20
	time	**300.20**	401.20	6480.20	11054.60
$R9^4$	iter / B&B nodes	1860.40	1853.70	40.20	133.70
	time	**312.90**	412.70	11870.00	37574.80
$R9^5$	iter / B&B nodes	1952.70	1910.70	123.40	377.30
	time	**344.50**	421.60	35831.70	105456.10
$R9^6$	iter / B&B nodes	2025.70	1962.00	200.80	867.90
	time	**349.60**	438.00	58707.70	269048.00
$R9^7$	iter / B&B nodes	2292.40	2136.00	465.00	2050.60
	time	**400.40**	475.90	138454.70	925843.40
$R9^8$	iter / B&B nodes	2313.60	2133.30	1141.60	3936.50
	time	**405.00**	484.60	341991.60	1984222.40
Average iter / B&B nodes		**1777.86**	**1444.11**	**251.92**	**1222.78**
Average time		**127.41**	**188.94**	**30333.77**	**823769.60**

Table 4.7: Comparison among the proposed solution approaches

4.5 Conclusions

In this paper, the elementary shortest path problem with forbidden paths has been addressed. The problem under study has been formulated as a specific instance of the resource constrained shortest path problem.

The single-origin single-destination and the single-origin all-destination versions of the problem have been handled and *B&B* and dynamic programming based solution approaches have been defined and implemented. Different versions of the two types of solution approaches have been developed.

Regarding to the *B&B* method, a naive and two enhanced versions have been defined. Both node and label selection versions of a dynamic programming based algorithm have been considered. In addition, several rules have been implemented to select, at each iteration, the label/node to be processed.

An extensive computational study has been carried out on a variety of network instances with the goal of assessing the behavior of the proposed solution procedures. With this aim, two groups of instances have been considered. The first one is characterized by a set of forbidden paths randomly generated , the second one is generated in such a way that all the forbidden paths are present as subsequence of the optimal solution of the relaxed problems (i.e., *B&B* node).

The collected numerical results underline that the performance of the proposed solution approaches is influenced by both the number of the additional constraints and the dimension of the problems that have to be solved.

For the test problems, in which the set of forbidden paths is randomly generated, the best performing versions of label and node selection methods outperform, on average, the best version of the *B&B* based solution approaches. This is observed for both the single-source single-destination and the single-source all-destination versions of the considered problem. However, it is worth observing that the B&B approach outperforms the dynamic programming based solution methods for the test problems of small size, i.e., low number of nodes and arcs.

The results, obtained on the instances of the second group, underline that the dynamic programming based methods remarkable outperform the best version of the $B\&B$ solution approaches.

In conclusion, the $B\&B$ strategies developed to solve the elementary shortest path problem with forbidden paths could be competitive with the dynamic programming solution approaches only if the number of violated constrains is very limited respect to the total number of forbidden paths and when the number of nodes and the number of arcs of the considered problem is not too high, i.e., networks with a number of nodes and a number of arcs less than or equal to 400 and to 5000, respectively. In addition, when the number of violated constrains increases, the computational cost of the $B\&B$ methods increases dramatically and they behave very poorly. On the contrary, the dynamic programming approaches seem to be very effective in solving the problem under study.

Chapter 5

Multi-dimensional labelling approaches to solve the linear fractional elementary shortest path problem with time windows [1]

Luigi Di Puglia Pugliese

Department of Electronics, Informatics and Systems, University of Calabria, 87036, Rende (CS), Italy

Francesca Guerriero

Department of Electronics, Informatics and Systems, University of Calabria, 87036, Rende (CS), Italy

Abstract

This paper investigates the linear fractional shortest path problem with time windows. For the specific problem, an elementary path with a minimum cost/time ratio is sought in a directed graph, where

[1]Accepted for pubblication in the journal *Optimization Methods & Software*, DOI: 10.1080/10556780903483364

two parameters (i.e. cost and time) are associated with each arc and a time window is associated with each node. Indeed, a valid path must satisfy the time window constraints, which are assumed to be of the hard type. Multi-dimensional labelling algorithms are proposed to solve this variant of the classical shortest path problem. Extensive computational tests are carried out on a meaningful number of test problems, with the goal of assessing the behaviour of the proposed approaches. The computational study shows that the introduction of dominance rules and the adoption of a bi-directional search strategy allow the definition of solution approaches that turn out to be very effective in solving the problem under consideration.

Keywords: shortest path problem; time window constraints; bi-criteria network; fractional objective function; multiple labelling approaches.

5.1 Introduction

The classical shortest path problem has been defined as a single-objective problem, in which the main objective is to optimize a single criterion, such as the total cost and the travel time [6].

However, in many real applications, arising in the design and management of different types of network (i.e. transportation, transmission, and communication networks), the complexity of the social and economic environment requires the explicit and simultaneous consideration of multiple and conflicting objectives ([6], [14], [33], [41], [42]).

In this framework, unless a well-defined utility function exists, it is not possible to identify a single optimal solution, but rather different paths exist (i.e. Pareto-optimal paths), which can be considered as best solutions in the sense that no improvement in any criterion is possible without sacrificing at least one of the other criteria.

Consequently, the decision-maker should be able to get a satisfactory compromise solution from the Pareto-optimal solution set. While the single objective shortest path problem has a polynomial complexity bound [20], its multi-criteria counterpart is classified as NP-complete [71]. Despite its theoretical complexity and given its practical importance, several studies have focused on the development of efficient solution approaches to address the multi-criteria shortest path problem. They can be classified into four groups. The first group includes exact procedures such as multiple labelling methods ([24], [35], [79], [103], [128]), ranking methods ([10], [36]) and parametric methods ([110]). The second group contains the approximate procedures ([82], [85], [134], [135], [142]). The third group of solution approaches utilizes the utility (or cost) functions ([27], [56], [107], [108]). The last group includes the interactive methods ([39], [40], [43], [74], [112]).

A review of the scientific literature underlines that, when only two attributes are taken into account, the main goal is generally to find a path for which the ratio of the two criteria is minimized. For example, in the work of Ahuja [3], the ratio of cost and reliability (i.e. the product of the reliabilities of the arcs in that path) is considered, whereas the papers of Martins [102] and Fox [67] address the minimum cost/time ratio path problem. The minimum cost/capacity ratio path problem has been studied in [101], and Sherali et al. [126] present a discrete fractional programming formulation to determine a path through a bi-attribute network, where each arc has a probability of accident and its consequence (cost) in order to minimize the risk of low probabilityhigh consequence accidents in transporting hazardous materials.

Recently, the general problem of identifying an optimal elementary path between a pair of nodes that minimizes the ratio of two linear functions has been studied by Soroush ([129], [130]).

In particular, in [129], an exact solution approach is proposed. The algorithm is based on an equivalent network model where a classical (single attribute) shortest path problem is repeatedly solved in order to get the solution for the bi-attribute shortest path problem. In [130], the author equivalently formulates the problem as a bi-attribute rational path problem and develops some path preference structures and elimination techniques to discard, from further consideration, those bi-attribute paths that cannot be a candidate for the optimal path.

The problem addressed here is the linear fractional elementary shortest path problem with time windows ($\mathcal{LFESPPTW}$, for short). This is an optimal path problem, defined through a bi-criteria network, with a fractional objective function and involving time window restrictions at the nodes. It is assumed that either arrivals and departures from each node are allowed only in time instants belonging to the corresponding time window and this constraint should not be violated under any circumstance. Indeed, the time window constraints are of the hard type.

Originally introduced by Roan and Lee [122], it consists of finding a path between a pair of nodes such that the cost per unit time is minimized and the time window constraints are satisfied. It is also required that no cycles can be a part of any solution.

The $\mathcal{LFESPPTW}$ is strongly related to the family of the elementary shortest path problem with resource constraints ($\mathcal{ESPPRC}$). Different solution approaches have been published in the scientific literature to address the $\mathcal{ESPPRC}$. We cite, for example, the work of Feillet et al. [64], in which a label correcting algorithm based on a node selection strategy, has been proposed to solve the $\mathcal{ESPPRC}$ in general graphs even if they contain negative cost cycles. In [64], the authors introduce the definition of unreachable nodes by considering the situation in which the triangle inequality holds for all the resources. They observe that sometimes it is possible to identify these kinds of nodes that cannot be visited in any feasible extension of a given state because of the resource limitation. Righini and Salani

[120] proposed a method alternative to the dynamic programming approach to solve the $\mathcal{ESPPRC}$. In particular, the authors consider a statespace relaxation and a branch and bound scheme is defined to close the gap between the upper and lower bound. Successively, the same authors developed a bi-directional dynamic programming approach, based on different dominance and bounding rules, to handle the $\mathcal{ESPPRC}$ ([119]). In all the aforementioned works, the strategy of considering a further dummy node resource, originally proposed by Beasley and Christofides ([16]), is adopted to avoid that the generated paths contain cycles. This idea has been considered also in [96], where a statespace relaxation of a dynamic programming algorithm for the $\mathcal{SPPRC}$ with node resources, where node resources are included only for some nodes, is proposed. The solution approaches presented in [96] have been tested computationally in [22], where the non-elementary version of the problem is solved iteratively until an elementary path is found and used. An extension of the concept of unreachable nodes presented in [64] is adapted to the case in which, for each considered resource, the triangle inequality is not satisfied. Indeed, they defined a node j as an unreachable node from a generic state, corresponding to node i, by considering a lower bound on the consumption of resource from node i to node j. Several alternative approaches to augmenting the node resources are defined and evaluated experimentally.

In this paper, we present optimal approaches to address the $\mathcal{LFESPPTW}$, that follows the solution strategies proposed to solve the $\mathcal{ESPPRC}$ in [64] and [119]. They are developed on the basis of the different label selection and node selection strategies, defined in such a way that the most promising node/label is selected at each iteration ([19]). In order to develop efficient solution approaches, a bi-directional search strategy that resembles the approach presented in [119] and some innovative dominance rules tailored to the problem under study, are also exploited. It is worth observing that, with reference to the classification of the procedures for the multi-criteria shortest path problem given above, the proposed approaches can be classified as belonging to the first group (i.e. exact procedures).

Our work has been motivated by various reasons. First of all, the current literature on the $\mathcal{LFESPPTW}$ indicates that scant attention has been given to the development of efficient approaches to deal with this problem. The only works that address the $\mathcal{LFESPPTW}$ are concerned with label-correcting methods (optimal and heuristic) based on the label selection strategy ([122]). These methods, in the choice of the label to be examined, do not take into account its value but instead they follow a breadth-first search of the network (i.e. FIFO label selection strategy); in addition, no dominance criteria are used to cut off unpromising labels ([64], [119]). On the other hand, to the best of our knowledge, the idea to select at each iteration a node instead of a label has not been considered. In addition, computational experiments have been carried out only on acyclic networks ([122]).

The rest of the present paper is organized as follows. In Section 5.2, we give the mathematical formulation of the $\mathcal{LFESPPTW}$. Section 5.3 presents the basic elements of the proposed solution approaches. The label selection methods developed to address the problem under investigation are presented in Section 5.4, whereas Section 5.5 is devoted to the description of the proposed node selection approaches. The effectiveness of the developed algorithms is tested on a large set of test problems and the related computational results are discussed in Section 5.6. Some concluding remarks are given in Section 5.7. The paper ends with a detailed account of the numerical results reported in Appendix 1 and with graphical representations of the steps executed by the proposed solution approaches, given in Appendix 2.

5.2 Mathematical formulation

The $\mathcal{LFESPPTW}$ is defined on a directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$, where $\mathcal{A}$ is the set of m arcs and $\mathcal{N}$ is the set of n nodes, including an origin node s and a destination node d.

For each node $i \in \mathcal{N}$, we denote by $FS(i) = \{j : (i,j) \in \mathcal{A}\}$ and $BS(i) = \{j : (j,i) \in \mathcal{A}\}$ the forward star and backward star of the node, respectively. Each arc $(i,j) \in \mathcal{A}$ has two parameters associated with it: a non-negative duration d_{ij}, which is the time required to travel from node i to node j, and a non-negative $cost_{ij}$. Each node $i \in \mathcal{N}$ has a non-negative duration d_i, a non-negative cost c_i and a time window$[a_i, b_i]$ associated with it. It is assumed that the time window constraints are of the hard type.

It is worth observing that it is not possible to wait at the start node otherwise a path could be cheaper by starting later or it could be the only feasible path. Thus, leaving from node s is allowed only at a time instant equal to zero. In other words, we impose that $a_s = b_s = 0$. In what follows, for each arc $(i,j) \in \mathcal{A}$, t_{ij} denotes the time needed to traverse the arc d_{ij} plus the duration d_i of node i (i.e., $t_{ij} = d_{ij} + d_i$), whereas c_{ij} represents the cost of traversing the arc $cost_{ij}$ plus the cost c_i associated with the node i (i.e., $c_{ij} = cost_{ij} + c_i$).

Given the presence of the time window constraints, each node $i \in \mathcal{N}$ must be visited within its time window $[a_i, b_i]$. Consequently, if node i is reached at the earliest possible time a_i, the node j may be visited before the end of its own time window and if node i is reached at the latest possible time b_i, then node j may be visited after the start of its own time window. Thus, an arc (i,j) is included into the set $\mathcal{A}$ only if it is feasible, that is, if the conditions $a_i + t_{ij} \leq bj$ and $b_i + t_{ij} \geq a_j$ are satisfied. Given two distinct nodes i and j, a path π_{ij} from node i to node j is a sequence of nodes $\pi_{ij} = \{i = i_1, \ldots, i_1 = j\}$, $l \geq 2$ such that $(i_k, i_{k+1}) \in \mathcal{A}$, for $k = 1, \ldots, l-1$. A path is *elementary* if it has no repeated occurrences of any node. A cycle C is a path with no repeated nodes, except the initial and terminal nodes that coincide, i.e., $i_1 \equiv i_l$. In what follows, we assume that the nodes visited along the path are all distinct, i.e., the path is elementary. A path π_{ij} from node i to node j is evaluated by the objective function $f(\pi_{ij}) = \sum_{\{(u,v) \in \pi_{ij}\}} c_{ij} / \sum_{\{(u,v) \in \pi_{ij}\}} t_{ij}$. Given the origin node s and the destination node d, the main aim of the $\mathcal{LFESPPTW}$ is to find an elementary path from s to d, for which the cost per unit time is minimized and the time window constraints, at each visited node, is satisfied.

The $\mathcal{LFESPPTW}$ can be mathematically formulated as follows.

$$\min \frac{\sum_{(i,j)\in\mathcal{A}} c_{ij}x_{ij}}{\sum_{(i,j)\in\mathcal{A}} t_{ij}x_{ij}} \tag{5.1}$$

$s.t.$

$$\sum_{\{j:(i,j)\in\mathcal{A}\}} x_{ij} - \sum_{\{j:(j,i)\in\mathcal{A}\}} x_{ji} = \begin{cases} 1 & \text{if } i = o \\ -1 & \text{if } i = d \\ 0 & otherwise \end{cases} \quad \forall i \in \mathcal{N} \tag{5.2}$$

$$x_{ij}(T_i + t_{ij} - T_j) = 0, \quad \forall (i,j) \in \mathcal{A} \tag{5.3}$$

$$a_i \le T_i \le b_i, \quad \forall (i,j) \in \mathcal{A} \tag{5.4}$$

$$x_{ij} \in \{0,1\}, \quad \forall (i,j) \in \mathcal{A} \tag{5.5}$$

where $T_i, \forall i \in \mathcal{N}$, represents the time of a feasible path from the origin node s to node i.

The mathematical model introduced above ensures that no waiting is allowed at each node and the nodes are visited in their time window. It is important to observe that if waiting at the nodes was allowed, then it would be possible to leave each node $i \in \mathcal{N}$ at the time b_i. In this situation, one could arrive to node d at a certain instant of time and wait up to b_d. This means that all the feasible solutions of the problem could be characterized by $T = b_d$, that is, the maximum time to arrive to node d. In this case, the problem could be addressed as a classical shortest path problem with time windows, and the related value of the fractional objective function would be the ratio between the least cost and T. On the contrary, in the problem under consideration, it is assumed that waiting at nodes is not allowed and all the paths from node s to node i, $\forall i \in \mathcal{N}$, that arrive to i either before a_i or after b_i are discarded. In other words, the departure time to node i is equal to the arrival time plus the duration of node i.

Since finding an elementary shortest path with time windows is a strongly NP-hard problem [[28],[59]], an obvious complexity result is reported in what follows.

Proposition 5.2.1. *$\mathcal{LFESPPTW}$, in general cases, belongs to the NP-hard class of problems.*

5.3 The proposed solution approaches

In the dynamic programming approaches proposed to solve the LFE-SPPTW, a set of labels is associated with each node, since many feasible paths may exist starting from s to every other node.

The hth label associated with node i, $i \in \mathcal{N}$ takes the following basic form, $y_i^{(h)} = (S_i^{(h)}, C_i^{(h)}, T_i^{(h)})$, and represents the main characteristics of the $h-th$ path $\pi_{si}^{(h)}$ from s to i.

In particular, the vector $S_i^{(h)} \in Z^n$ is used to store the number of times in which each node is visited along the path $\pi_{si}^{(h)}$ ([16], [64], [119], [120]). In our problem, feasible paths are not allowed to contain cycles, thus each element of S has to be less than or equal to one.

$C_i^{(h)}$ represents the total cost of the path π_{si}^h (i.e., $C_i^{(h)} = \sum_{(i,j)\in\pi_{si}^{(h)}} c_{ij}$), whereas $T_i^{(h)}$ denotes the total time associated with $\pi_{si}^{(h)}$, that is, $T_i^{(h)} = \sum_{(i,j)\in\pi_{si}^{(h)}} t_{ij}$. It is worth observing that, depending on the specific version of the considered solution approach, additional information has to be stored in the labels associated with each node. This aspect will be better discussed in the following sections.

On the basis of the definitions introduced above, it is evident that the $h-th$ path $\pi_{si}^{(h)}$ from s to i associated with label $y_i^{(h)} =$

$(S_i^{(h)}, C_i^{(h)}, T_i^{(h)})$ is feasible only if the following conditions are satisfied:

$$\begin{cases} S_i^{(h)}[j] = 0, 1, \quad j = 1, \dots, n \\ a_i \leq T_i \leq b_i. \end{cases} \tag{5.6}$$

In addition, when a feasible path $\pi_{si}^{(h)}$ is extended along the arc $(i, j) \in \mathcal{A}$ to the path $\pi_{sj}^{(k)}$, the newly created label $y_j^{(k)} = (s_j^{(k)}, C_j^{(k)}, T_j^{(k)})$ is determined on the basis of the following updating rules:

$$S_j^{(k)}[j] = \begin{cases} S_i^{(h)} + 1 & \text{if } n_l = j \\ S_i^{(h)} & \text{if } n_l \neq j \end{cases}, \quad n_l = 1, \dots, n;$$
$$C_j^{(k)} = C_i^{(h)} + c_{ij}; \quad T_j^{(k)} = T_i^{(h)} + t_{ij}.$$

The labels associated with each node are determined iteratively during the execution of the algorithm and are stored in the set $D(i) = \{y_i^{(1)}, \dots, y_i^{(l)}\}$, $\forall i \in \mathcal{N}$.

The solution approaches developed to address the $\mathcal{LFESPPTW}$ can be viewed as an extension of the node selection and label selection procedures developed in [76] for solving the multi-criteria shortest path problem. In the first case, at each iteration, a node i is selected and all the labels associated with it are used to update the set of labels of the nodes adjacent to i. In a label selection method, at each iteration, a label is selected and a new label is determined for each adjacent node j. Different versions of these two procedures can be defined by considering different node and label selection strategies. In addition, depending on the way the labels are extended, we can define three different variants of the label selection and node selection procedures: the forward variant, where the labels are extended forward from the source node; the backward variant, in which the labels are extended backward from the destination node; and the bi-directional variant, where the labels are iteratively extended forward from node s and backward from node d.

5.3.1 Label fathoming rules

The effectiveness of the proposed labelling approaches strongly depends on the possibility of fathoming labels/paths that cannot lead to an optimal solution. In order to achieve this goal, three different rules have been developed: a lower bound rule and two dominance rules.

The first one resembles the idea proposed in [22]. Indeed, in order to reduce the number of generated labels, a lower bound is used to fathom partial paths that do not allow the determination of a complete feasible path from the origin node s to the destination node d. In particular, for each pair of nodes $i, j \in \mathcal{N}$, the lower bound LB_{ij} represents the value of the minimum time path from i to j. LB_{ij} is also used to identify the unreachable nodes, as described in [22] and to update coherently the vector S.

Two dominance rules have also been introduced to cut off unpromising labels. The first dominance test deals with the well-known concepts of Pareto optimality when tailored for the LFESPPTW and can be stated as follows.

Definition 5.3.1. *Let $\pi_{si}^{(1)}$ and $\pi_{si}^{(2)}$ be two different paths from node s to node i with associated labels $y_i^{(1)} = (S_i^{(1)}, C_i^{(1)}, T_i^{(1)})$ and $y_i^{(2)} = (S_i^{(2)}, C_i^{(2)}, T_i^{(2)})$. Then the former dominates the latter if $T_i^{(1)} = T_i^{(2)}$, $C_i^{(1)} < C_i^{(2)}$ and $S_i^{(1)} < S_i^{(2)}$*

The definition of the equivalence between labels is given in what follows.

Definition 5.3.2. *Let $\pi_{si}^{(1)}$ and $\pi_{si}^{(2)}$ be two different paths from node s to node i with associated labels $y_i^{(1)} = (S_i^{(1)}, C_i^{(1)}, T_i^{(1)})$ and $y_i^{(2)} =$*

$(S_i^{(2)}, C_i^{(2)}, T_i^{(2)})$. Then $\pi_{si}^{(1)}$ is equivalent to $\pi_{si}^{(2)}$ if $T_i^{(1)} = T_i^{(2)}$ and $C_i^{(1)} = C_i^{(2)}$.

In the proposed approaches, no equivalent labels are generated.

The paper of Ioachin et al. [88] has served as a departure point for designing the second dominance rule proposed here, which exploits the specific structure of the objective function and is valid only for acyclic networks. The main idea is reported in the following.

Assume that two labels $y_i^{(h)}$ and $y_i^{(k)}$ are associated with a generic node i and they verify the following relation $f(\pi_{si}^{(h)}) \geq f(\pi_{si}^{(k)})$. The question is: 'Is it possible to find a condition that guarantees that, by extending these labels to a node $j \in FS(i)$, we obtain two paths $\pi_{sj}^{(h)}$ and $\pi_{sj}^{(k)}$ for which the relation $f(\pi_{sj}^{(h)}) \geq f(\pi_{sj}^{(k)})$ is also satisfied'? In the affirmative case, by applying iteratively the reasoning outlined above, it is possible to conclude that the aforementioned relations must hold also for the successors of node j and so on. Consequently, if, when starting from node i it is possible to reach the destination node d by a feasible path, then we obtain the paths from s to d by extending iteratively the two paths $y_i^{(h)}$ and $y_i^{(k)}$ that satisfy the relation $f(\pi_{sd}^{(h)}) \geq f(\pi_{sd}^{(k)})$ and, thus, the path $y_i^{(h)}$ can be discarded because an equivalent or an alternative path from s to d with a smaller objective function value can be determined starting from the partial path $y_i^{(k)}$.

The conditions that guarantee the satisfaction of the relations introduced above are formally stated in the following theorem.

Theorem 5.3.3. *Let $y_i^{(1)}, y_i^{(2)} \in D(i)$ be two different labels associated with node i such that*

$$\frac{C_i^{(1)}}{T_i^{(1)}} \leq \frac{C_i^{(2)}}{T_i^{(2)}} \tag{5.7}$$

Assume the extension of these labels to node $j \in FS(i)$. It is possible *to show that*

$$\frac{C_i^{(1)} + c_{ij}}{T_i^{(1)} + t_{ij}} \leq \frac{C_i^{(2)} + c_{ij}}{T_i^{(2)} + t_{ij}} \tag{5.8}$$

only if $C_i^{(1)} - C_i^{(2)} \leq 0$ *and* $T_i^{(2)} - T_i^{(1)} \leq 0$.

Proof. Condition (5.8) can be rewritten as follows:

$$\begin{aligned}
(C_i^{(1)} + c_{ij})(T_i^{(2)} + t_{ij}) \leq (C_i^{(2)} + c_{ij})(T_i^{(1)} + t_{ij});\\
C_i^{(1)}T_i^{(2)} + t_{ij}C_i^{(1)} + c_{ij}T_i^{(2)} + c_{ij}t_{ij} \leq\\
\leq C_i^{(2)}T_i^{(1)} + t_{ij}C_i^{(2)} + c_{ij}T_i^{(1)} + c_{ij}t_{ij};
\end{aligned}$$

$$C_i^{(1)}T_i^{(2)} - C_i^{(2)}T_i^{(1)} + t_{ij}(C_i^{(1)} - C_i^{(2)}) + c_{ij}(T_i^{(2)} - T_i^{(1)}) \leq 0. \tag{5.9}$$

Condition (5.7) implies that $C_i^{(1)}T_i^{(2)} - C_i^{(2)}T_i^{(1)} \leq 0$. Being $t_{ij} \geq 0$ and $c_{ij} \geq 0$, (5.9) is valid only if the following is satisfied:

$$C_i^{(1)} - C_i^{(2)} \leq 0; \quad T_i^{(2)} - T_i^{(1)} \leq 0.$$

Assume the extension of the labels $y_i^{(1)}, y_i^{(2)} \in D(i)$ along a generic feasible path π_{id} from node i to the destination node d. Condition (5.9) assumes the following form:

$$C_i^{(1)}T_i^{(2)} - C_i^{(2)}T_i^{(1)} + \sum_{(i,j)\in\pi_{id}} t_{ij}(C_i^{(1)} - C_i^{(2)}) + \sum_{(i,j)\in\pi_{id}} c_{ij}(T_i^{(2)} - T_i^{(1)}) \leq 0. \tag{5.10}$$

Similarly to the previous case, since $\sum_{(i,j)\in\pi_{id}} t_{ij} \geq 0$ and $\sum_{(i,j)\in\pi_{id}} c_{ij} \geq 0$, the condition (5.10) is valid only if $C_i^{(1)} - C_i^{(2)} \leq 0$ and $T_i^{(2)} - T_i^{(1)} \leq 0$. □

In Theorem 5.3.3, a generic feasible path π_{id} starting from node i and ending at node d is assumed to exist. This means that two labels can be compared only if a feasible sequence of arcs, that completes the corresponding partial paths, exists for both the labels $y_i^{(1)}, y_i^{(2)} \in D(i)$. For each node $i \in \mathcal{N}$ it is possible to define a new time window $[a_i^{'}, b_i^{'}]$ in which it is possible to leave, and we are sure that at least one feasible extension up to the destination node d exists for both labels.

It is important to point out that a procedure to reduce the width of the time windows has also been proposed in [51]. The authors compute a new time window for each node with a smaller feasible range than the original one; this leads to an improvement in the performance of the algorithm without compromising the optimality of the solution determined. The procedure removes, for each node, the time instants for which no feasible paths to the destination node exist.

The goal of the pre-processing strategy proposed here is different. Indeed, for each node $i \in \mathcal{N}$, the new defined range $[a_i^{'}, b_i^{'}]$ is determined in such a way that all the labels belonging to it can be used to determine complete feasible paths reaching the destination node by considering all the possible arc sequences that begin at node i. In addition, nothing can be said for the labels that are not included in this interval, and thus these labels cannot be considered when applying the second dominance rule.

To compute $[a_i^{'}, b_i^{'}]$, a backward pre-processing procedure has been devised. In particular, starting from node d, the new time windows are defined by letting: $a_d^{'} = a_d$; $b_d^{'} = b_d$; $a_i^{'} = 0$, $b_i^{'} = +\infty$, $\forall i \in \mathcal{N} - \{d\}$; $a_i^{'} = \max\{a_i; a_i^{'}; \max_{j:(i,j)\in\mathcal{A}} a_j^{'} - t_{ij}\}$ and $a_i^{'} = \min\{b_i; b_i^{'}; \min_{j:(i,j)\in\mathcal{A}} b_j^{'} - t_{ij}\}$.

This procedure does not ensure that $b_i^{'} \geq a_i^{'}$. If $b_i^{'} < a_i^{'}$ we impose that $b_i^{'} = a_i^{'} = 0$. In the case in which $b_i^{'} \geq b_i$ and $a_i^{'} \leq a_i$ we assume that $[a_i^{'}; b_i^{'}] \equiv [a_i; b_i]$.

The pre-processing strategy outlined above is valid only for acyclic networks. Indeed, the recursive nature of the procedure itself prevents us from applying it to cyclic networks. Indeed, the interval $[a_i^{'}; b_i^{'}]$ cannot be correctly defined, since the presence of a cycle implies that the values of $a_i^{'}$ and $b_i^{'}$ tend to zero. In addition, it can be applied only in the forward version of the proposed labelling approaches, in which the labels are extended forward from the source node.

When the labels are extended backward from the destination node, the new time windows are determined by applying a forward pre-processing procedure, which requires the execution of the following operations: $a_s^{'} = a_s$; $b_s^{'} = b_s$; $a_j^{'} = 0$, $b_j^{'} = +\infty$, $\forall j \in \mathcal{N} - \{o\}$; $a_j^{'} = \max\{a_j^{'}; \max_{i:(i,j)\in\mathcal{A}} a_i^{'} + t_{ij}\}$ and $b_j^{'} = \min\{b_j^{'}; \min_{i:(i,j)\in\mathcal{A}} b_i^{'} + t_{ij}\}$.

The theoretical results stated in Theorem 5.3.3 suggest a new dominance criterion that is valid only for acyclic networks. For this reason, the dummy node resources are not considered, and thus only the time and the cost values are used to compare two different paths ending at the same node.

Definition 5.3.4. *Let $\pi_{si}^{(1)}$ and $\pi_{si}^{(2)}$ be two different paths from node s to node i with associated labels $y_i^{(1)} = (C_i^{(1)}, T_i^{(1)})$ and $y_i^{(2)} = (C_i^{(2)}, T_i^{(2)})$, where $T_i^{(1)})$ and $T_i^{(2)}) \in [a_i^{'}; b_i^{'}]$. Then $\pi_{si}^{(1)}$ (or $(C_i^{(1)}, T_i^{(1)})$) dominates $\pi_{si}^{(2)}$ (or $(C_i^{(2)}, T_i^{(2)})$) if and only if $f(\pi_{si}^{(1)}) \leq f(\pi_{si}^{(2)})$, $C_i^{(1)} - C_i^{(2)} \leq 0$, $T_i^{(2)} - T_i^{(1)} \leq 0$.*

It is evident from Definition 5.3.4, that the second dominance criterion can compare also two equivalent labels. In this case, the latter is discarded. Of course, this does not influence the correctness of the related approach and is coherent with the aim of not generating equivalent paths. In addition, this second rule contains the first dominance criterion as a special case, stated in Definition 5.3.1.

5.4 Label selection methods

In a label selection approach, at each iteration a label is selected and processed. The main step is represented by the scanning of a label, with the aim to create new labels. In particular, scanning the label $y_i^{(k)}$ associated with node i involves the examination of all the adjacent nodes j of i (i.e. either $j \in FS(i)$ or $j \in BS(i)$) and the determination of a new label $\bar{y}_j$. The new label is then compared to all the existing labels of node j, that is, the labels belonging to $D(j)$. If $\bar{y}_j$ is not dominated, then it is added to $D(j)$ and all the labels in $D(j)$ dominated by $\bar{y}_j$ are discarded.

In the following sections, we provide a detailed description of the forward version (referred to as Forward Label Selection Method ($\mathcal{FLSM}$), in which the labels are extended forward from the source node of the backward version (referred to as Backward Label Selection Method ($\mathcal{BLSM}$), where the labels are extended backward from the destination node, and of the bi-directional variant (referred to as the Bi-Directional Label Selection Method ($\mathcal{BDLSM}$), where the labels are iteratively extended forward from node s and backward from node d.

5.4.1 Forward label selection method

The $\mathcal{FLSM}$ maintains a candidate list L, storing all the labels with the potential to determine a new label for at least one node. At each iteration, a label $y_i^{(k)}$ is selected and a new label is determined for all successors of node i. The algorithm terminates when the list L is found to be empty, that is, there are no more labels to be scanned. In the proposed approach, the lower bound rule is applied. Consequently, when a label $y_i^{(k)}$ is selected, a new label $y_j^{(h)}$ for the adjacent node j is determined only if the conditions $S_i^{(k)}[j] = 0, a_j \leq T_i^{(k)} + t_{ij} \leq b_j$ and $T_i^{(k)} + t_{ij} + LB_{jd} \leq b_d$ are satisfied. In addition, the lower bound is used to identify the unreachable nodes, that is, the nodes that cannot be reached, starting from the newly created partial path $y_j^{(h)}$. In particular, a node $v \in \mathcal{N}$ is unreachable from j if $T_i^{(k)} + t_{ij} + LB_{jv} > b_v$. In order to keep trace of these nodes, the vector S is updated coherently. In particular, we set $S_j^{(h)}[v] = 1, \forall v \in \{\mathcal{N} : T_i^{(k)} + t_{ij} + LB_{jv} > b_v\}$. The main operations executed by the $\mathcal{FLSM}$ can be formally stated as described in Algorithm 11.

Algorithm 11 $\mathcal{FLSM}$ Scheme

```
Step 0 (Initialization phase)
Set: $D(s) = \{y_s^{(1)}\}$, with $S_s^{(1)}[s] = 1, S_s^{(1)}[h] = 0 \;\; \forall h \in \mathcal{N} - \{s\}, T_s^{(1)} = C_s^{(1)} = 0; L = \{y_s^{(1)}\}$.

Step 1 (Label selection)
if $L = \emptyset$ then
    STOP.
else
    Select a label $y_i^{(m)} = (S_i^{(m)}, C_i^{(m)}, T_i^{(m)})$ of $L$ and delate it from $L$.
end if

Step 2 (Label scan)
for all $j \in FS(i)$ do
    if $S_i^{(m)}[j] = 0$ AND $a_j \le T_i^{(m)} + t_{ij} \le b_j$ AND $T_i^{(m)} + t_{ij} + LB_{jd} \le b_d$ then
        Set: $\bar{T}_j = T_i^{(m)} + t_{ij}; \bar{C}_j = C_i^{(m)} + c_{ij}$;
        $\bar{S}_j[w] = S_i^{(m)}[w], \forall w \in \mathcal{N} - \{\{j\} \cup \{l : \bar{T}_j + LB_{jl} > b_l\}\}$.
        $\bar{S}_j[v] = 1, \forall v \in \mathcal{N}, v \neq w$.
        if $(\bar{S}_j, \bar{C}_j, \bar{T}_j)$ is dominated by some label belonging to $D(j)$ then
            Go to Step 1.
        else
            Set: $y_j^{(|D(j)|+1)} = (\bar{S}_j, \bar{C}_j, \bar{T}_j); D(j) = D(j) \cup \{y_j^{(|D(j)|+1)}\}; L = L \cup \{y_j^{(|D(j)|+1)}\}$.
            Remove from $D(j)$ and $L$ all the label that are dominated by $y_j^{(|D(j)|+1)}$.
        end if
    end if
end for
Go to Step 1.
```

5.4.2 Backward label selection method

In the $\mathcal{BLSM}$, the research process starts from the destination node d and paths ending at node s are determined. Also in this version of the algorithm, a list L of candidate labels is considered. At each iteration, a label $y_j^{(h)}$ is selected and a new label $y_i^{(k)}$ is obtained for all predecessors i of node j. The algorithm terminates when the list L is found to be empty.

As mentioned in Section 5.2, it is possible to leave from node s only at time zero. Thus, a backward path is feasible only if it ends up in s at time zero. On the other hand, node d can be reached at each instant time belonging to [ad, bd]. Consequently, an initial label, for each time unit at the ending time window [ad, bd] is associated with the destination node d and the value of the corresponding time instant is added to the related label.

Indeed, in the initialization phase, a set of $b_d - a_d + 1$ labels are associated with node d and the generic $k - th$ label $y_d^{(k)}$ takes the following form $y_d^{(k)} = (S_d^{(k)}, C_d^{(k)}, T_d^{(k)}, t_d^{(k)})$, with $S_d^{(k)}[d] = 1, S_d^{(k)}[h] = 0, \forall h \in \mathcal{N} - \{d\}, T_d^{(k)} = t_d^{(k)}, C_d^{(k)} = 0, t_d^{(k)} = k-1, k = 1, \ldots, b_d - a_d + 1$. Each label $y_d^{(k)}$ is a candidate to represent a path that starts at node s at time zero and arrives at node d attime $b_d - t_d^{(k)}$.

It is important to point out that it is possible to reduce the number of initial labels associated with node d, since node d should be reached at least at the time $t = \max\{a_d; \min_{i \in BS(d)}\{a_i + t_{id}\}\}$ and at most at the time $t = \min\{b_d; \max_{i \in BS(d)}\{b_i + t_{id}\}\}$.

The main step of the $\mathcal{BLSM}$ is represented by the extension of the label $y_j^{(h)}$. The newly created label $y_i^{(k)} = (S_i^{(k)}, C_i^{(k)}, T_i^{(k)}, t_i^{(k)}), i \in BS(j)$, is determined by setting: $S_i^{(k)}[t] = S_j^{(h)}[t], t \in \mathcal{N} - \{j\}, S_i^{(k)}[j] = 1; T_i^{(k)} = T_j^{(h)} + t_{ij}; C_i^{(k)} = C_j^{(h)} + c_{ij}; t_i^{(k)} = t_j^{(h)}$.

When a complete path $\pi_{ds}^{(w)}$, with relative label $y_s^{(w)} = (S_s^{(w)}, C_s^{(w)}, T_s^{(w)}, t_s^{(w)})$, at node s is found, the time needed to reach node d starting from node s at time 0 is equal to $T_s^{(w)} - t_s^{(w)}$.

It is important to point out that, for each arc $(i, j) \in \mathcal{A}$, the label $y_j^{(h)}$ is extended to node i only if the following conditions are satisfied [119].

$$\begin{cases} T - b_i \leq T_j + t_{ij} \leq T - a_i \\ S_j[i] = 0 \end{cases} \tag{5.11}$$

where the parameter T represents the maximum time in which a feasible path can arrive to node d and the node is served. For this reason, we have $T = b_d$ [119]. The value $T_j + t_{ij}$ represents the feasible arrival time to node i starting from node d, across node j.

It is possible to show that conditions (5.11) and the feasibility conditions (5.6), introduced for the $\mathcal{FLSM}$, are equivalent. Indeed, the following theoretical results hold.

Lemma 5.4.1. *The feasibility conditions (5.11) defined for the $\mathcal{BLSM}$ are equivalent to the feasibility conditions (5.6) introduced for the $\mathcal{FLSM}$.*

Proof. Let us consider the two inequalities:

$$T_j + t_{ij} \leq T - a_i, \tag{5.12}$$

$$T - b_i \leq T_j + t_{ij}. \tag{5.13}$$

Developing the inequality given in (5.12) we have

$$\begin{aligned} T_j + t_{ij} - T &\leq a_i; \\ -T + (T_j + t_{ij}) &\leq -a_i; \\ T - (T_j + t_{ij}) &\geq a_i. \end{aligned}$$

On the other hand, developing the inequality given in (5.13) we have the following:

$$\begin{aligned} -b_i &\leq -T + T_j + t_{ij}; \\ b_i &\geq T - (T_j + t_{ij}); \end{aligned}$$

where $T - (T_j + t_{ij})$ represents the arrival time to node i starting from node s at time zero.

Let T_i denote the arrival time to node i starting from node s at time zero, we have that

$$a_i \leq T_i \leq b_i \equiv T - b_i \leq T_j + t_{ij} \leq T - a_i.$$

This completes the proof of the Lemma. $\square$

The results of Lemma 5.4.1 allow us to use the conditions (5.11) as a feasibility check for the labels generated by the backward procedure. The lower bound rule introduced for the $\mathcal{FLSM}$ has also been extended to the backward counterpart. The main operations executed by the $\mathcal{BLSM}$ are reported in Algorithm 12.

Algorithm 12 $\mathcal{BLSM}$ Scheme

```
Step 0 (Initialization phase)
Set: $D(d) = \{y_d^{(1)}, y_d^{(2)}, \ldots, y_d^{(b_d-a_d+1)}\}$, with $S_d^{(k)}[d] = 1, S_d^{(k)}[h] = 0 \quad \forall h \in \mathcal{N} - \{d\}, T_d^{(k)} = t_d^{(k)}, C_d^{(d)} = 0, t_d^{(k)} = k-1, k = 1, \ldots, b_d - a_d + 1; L = \{y_d^{(1)}, y_d^{(2)}, \ldots, y_d^{(b_d-a_d+1)}\}$.

Step 1 (Label selection)
if $L = \emptyset$ then
    STOP.
else
    Select a label $y_j^{(m)} = (S_j^{(m)}, C_j^{(m)}, T_j^{(m)}, t_j^{(m)})$ of $L$ and delate it from $L$.
end if

Step 2 (Label scan)
for all $i \in BS(j)$ do
    if $S_j^{(m)}[i] = 0$ AND $T - b_i \leq T_j^{(m)} + t_{ij} \leq T - a_i$ AND $T_j^{(m)} + t_{ij} + LB_{si} \leq b_d$ then
        Set: $\bar{T}_i = T_j^{(m)} + t_{ij}; \bar{C}_i = C_j^{(m)} + c_{ij}; \bar{t}_i = t_j^{(m)}$;
        $\bar{S}_i[w] = S_j^{(m)}[w], \forall w \in \mathcal{N} - \{\{i\} \cup \{l : \bar{T}_i + LB_{li} > T - a_l\}\}$.
        $\bar{S}_i[v] = 1, \forall v \in \mathcal{N}, v \neq w$.
        if $(\bar{S}_i, \bar{C}_i, \bar{T}_i, \bar{t}_i)$ is dominated by some label belonging to $D(i)$ then
            Go to Step 1.
        else
            Set: $y_i^{(|D(i)|+1)} = (\bar{S}_i, \bar{C}_i, \bar{T}_i, \bar{t}_i); D(i) = D(i) \cup \{y_i^{(|D(i)|+1)}\}; L = L \cup \{y_i^{(|D(i)|+1)}\}$.
            Remove from $D(i)$ and $L$ all the label that are dominated by $y_i^{(|D(i)|+1)}$.
        end if
    end if
end for
Go to Step 1.
```

5.4.3 Bi-directional label selection method

A bi-directional approach, in which the labels are extended both forward from the source node to its successors and backward from the destination node to its predecessors, can outperform the $\mathcal{FLSM}$ outlined above [119]. For this reason, we developed a bounded bi-directional approach for the $\mathcal{LFESPPTW}$, which resembles the approach proposed by Righini and Salani for solving the $\mathcal{ESPPRC}$ [119]. As the procedure presented in [119], the proposed method can be viewed as characterized by the execution of three main steps. In the first step, the labels are extended forward from the source node without creating labels with the time component greater than half of the maximum allowed time. In the second step, similarly to the operations executed in the first step, the labels are extended backward from the destination node d. In the third and last phase, pairs of forward and backward labels, associated with the same node, are joined together to obtain a path from node s to node d. Infeasible paths are discarded.

The bi-directional version uses two lists of candidate labels, that is L^{fw} and L^{bw}, in which the most promising labels determined during the forward and the backward phase, respectively, are stored. In addition, for each node $i \in N$, the labels set $D(i)$ is viewed as the union of two sets, that is $D(i) = D^{fw}(i) \cup D^{bw}(i)$, where $D^{fw}(i)$ contains the labels generated during the forward phase (i.e. forward labels) and $D^{bw}(i)$ those determined during the backward step (i.e. backward labels). The algorithm terminates when the set $L^{fw} \cup L^{bw}$ is found to be empty. At this point, the optimal path is obtained by executing the $JOIN$ procedure, whose aim is to obtain a complete path from node s to node d by merging two partial paths. The related path must be feasible so as to satisfy the time-window constraints and it has to be an elementary path.

In particular, let $\pi_{si}^{(h)}$ be the partial feasible path corresponding to the label $y_i^{(h)} \in D^{fw}(i), h = 1, \dots, |D^{fw}(i)|$ and let $\pi_{id}^{(k)}$ be the partial feasible path associated with the label $y_i^{(k)} \in D^{bw}(i), k = 1, \dots, |D^{bw}(i)|$. For each node $i \in N$, it is possible to generate complete feasible paths $\pi_{sd}^{(w)} = \pi_{si}^{(h)} \cup \pi_{id}^{(k)}, h = 1, \dots, |D^{fw}(i)|; k = 1, \dots, |D^{bw}(i)|$, only if these complete paths are feasible (i.e. satisfy time window constraints and do not contain repeated nodes).

For each node i, the value $T_i^{(k)}$ of the $k-th$ label $y_i^{(k)} \in D^{bw}(i)$, determined during the backward phase, represents the arrival time to node i starting from node d. Consequently, on the basis of the conditions (5.11), these labels represent the partial paths, starting from d at time $T - t_i^{(k)} = b_d - t_i^{(k)}$.

The JOIN between two partial paths $\pi_{si}^{(h)}$ and $\pi_{id}^{(k)}$ associated with the labels $y_i^{(h)} \in D^{fw}(i)$ and $y_i^{(k)} \in D^{bw}(i)$, respectively, generates a feasible path $\pi_{sd}^{(w)} = \pi_{si}^{(h)} \cup \pi_{id}^{(k)}$ only if the following conditions are satisfied:

$$T_i^{(h)} = T - T_i^{(k)}; \tag{5.14}$$

$$S_i^{(h)}[t] + S_i^{(k)}[t] \leq 1 \quad \forall t \in \mathcal{N}, t \notin \{unrN_i^{(h)} \cup unrN_i^{(k)}\}; \tag{5.15}$$

where $unrN_i^{(h)}$ and $unrN_i^{(k)}$ represent the sets of unreachable nodes associated with the label $y_i^{(h)}$ and $y_i^{(k)}$, respectively.

It is worth observing that in conditions (5.15), the unreachable nodes are not taken into account. This guarantees that no feasible solution is lost. It is possible to show that the $JOIN$ procedure determines feasible paths. Indeed, the following result holds.

Theorem 5.4.2. *The $JOIN$ procedure generates feasible paths.*

Proof. First of all, it is worth observing that, given the labels $y_i^{(h)} \in D^{fw}(i)$ and $y_i^{(k)} \in D^{bw}(i)$, corresponding to the paths $\pi_{si}^{(h)}$ and $\pi_{id}^{(k)}$, respectively, the quantity $T_i^{(h)} + T_i^{(k)} - t_i^{(k)}$ represents the time required to reach node d, starting from node s at time zero, along the path obtained by merging the paths $\pi_{si}^{(h)}$ and $\pi_{id}^{(k)}$. Consequently, in order to ensure that the corresponding path is feasible (i.e. the time window constraints are fulfilled), the condition $T_i^{(h)} + T_i^{(k)} - t_i^{(k)} \leq T$, where T represents the latest arrival time to node d, must be satisfied. On the basis of (5.14), we can rewrite the previous condition as

$T - t_i^{(k)} \leq T$. Since $t_i^{(k)} \geq 0$, $T - t_i^{(k)} \leq T$ is always satisfied. This justifies the equality (5.14). Besides the satisfaction of the time window constraints, the obtained path must be elementary. In otherwords, the conditions $S_i^{(h)}[t] + S_i^{(k)}[t] \leq 1 \;\; \forall t \in \mathcal{N}, t \notin \{unrN_i^{(h)} \cup unrN_i^{(k)}\}$ have to be met. Then the relations (5.15). This completes the proof of the Theorem. □

To reduce the number of labels generated and to stop the extension of forward and backward paths while ensuring that the optimal solution will be found, bounding is used. The main idea is that we can stop extending a path in one direction when we have the guarantee that the remaining part of the path will be generated in the other direction and therefore no optimal solution will be lost.

It is worth observing that, without bounding, the bi-directional algorithm determines twice as many labels compared with either the forward or the backward versions. The bounding technique used for fathoming unpromising labels considers the time as a critical resource [119]. In particular, a label is extended only if the corresponding time is less than or equal to $T_{max}/2$, where T_{max} represents the maximum arrival time to node d (i.e., $T_{max} = b_d$). The rationale of this definition can be understood by taking into account that the condition $T_i^{(h)} + T_i^{(k)} - t_i^{(k)} \leq T$ with $y_i^{(h)} \in D^{fw}(i)$ and $y_i^{(k)} \in D^{bw}(i)$, where T denotes the maximum arrival time at node d (i.e., $T \equiv T_{max}$), should be satisfied. The value $T_i^{(w)} = T_i^{(h)} + T_i^{(k)} - t_i^{(k)}$ represents the time required to reach node d starting from node s at time 0.

It is evident that the time window constraint associated with node d is satisfied only if $T_i^{(w)} \leq T_{max}$. Consequently, the conditions $T_i^{(h)} \leq T_{max}/2$ and $T_i^{(k)} - t_i^{(k)} \leq T_{max}/2$ should be satisfied. In addition, in order to reduce the number of generated labels, by guaranteeing that the optimal path is found, it is possible to reduce the set of nodes to be considered when a label is selected.

In particular, in the forward phase [or backward phase], when a label $y_i^{(m)}$ [or $y_j^{(m)}$] is chosen and processed, a subset $\tilde{FS}(i)$ [or $\tilde{BS}(j)$] of the adjacent nodes, belonging to the forward [or backward] star is considered. Indeed, for each arc $(i, j) \in \mathcal{A}$, the following conditions must be satisfied:

$$\begin{aligned} &\forall y_i^{(m)} \in D^{fw}(i) \; \exists \; y_j^{(h)} \leftarrow y_i^{(m)} \oplus (i,j) \in D^{fw}(j) \\ &\leftrightarrow \nexists \; y_i^{(k)} \in D^{bw}(j) : y_i^{(k)} \leftarrow y_j^{(n)} \ominus (i,j), \; \forall y_j^{(n)} \in D^{bw}(j). \end{aligned} \tag{5.16}$$

where $\oplus$ represents the operator that extends forward the label $y_i^{(m)}$ along the arc (i, j) and $\ominus$ is the operator that extends backward the label $y_j^{(n)}$ along the arc (i, j).

The subsets $\tilde{FS}(i)$ and $\tilde{BS}(i)$ are defined as follows:

$$\begin{aligned} \tilde{FS}(i) &= \{j : (i,j) \in \mathcal{A}, \nexists y_i \in D^{bw}(i) : y_i \leftarrow y_j \ominus (i,j)\} \\ \tilde{BS}(i) &= \{k : (k,i) \in \mathcal{A}, \nexists y_i \in D^{fw}(i) : y_i \leftarrow y_k \oplus (i,j)\} \end{aligned}$$

Indeed, the set $\tilde{FS}(i)$ [or $\tilde{BS}(i)$] does not contain the node j [or the node k] if the set $D^{bw}(i)$ [or $D^{fw}(i)$] contains at least a label $y_i^{(m)} \leftarrow y_j^{(h)} \ominus (i, j)$ [or $y_i^{(m)} \leftarrow y_k^{(h)} \oplus (k, i)$].

The following theoretical results hold.

Theorem 5.4.3. *Replacing, for each node $i \in \mathcal{N}$, the set $FS(i)$ [or $BS(i)$] with the set $\tilde{FS}(i)$ [or $\tilde{BS}(i)$] does not compromise the determination of the optimal solution.*

(1) The paths generated by the $JOIN$ procedure, based on conditions (5.16) are all different.

(2) Using the conditions (5.16), all feasible paths from s to d are determined.

Proof. (1) Suppose that the $JOIN$ procedure is applied to a generic node i. For each $y_i^{(m)} \in D^{fw}(i)$ and $y_i^{(l)} \in D^{bw}(i)$, if conditions (5.14)-

(5.15) are verified, the $JOIN$ procedure generates a complete path $\pi_{sd}^{(w)} = \pi_{si}^{(m)} \cup \pi_{id}^{(l)}$ with the associated label $y_d^{(w)} = (S_d^{(w)}, C_d^{(w)}, T_d^{(w)}) = (S_i^{(m)} \cup S_i^{(l)}, C_i^{(m)} + C_i^{(l)}, T_i^{(m)} + T_i^{(l)} - t_i^{(l)})$. If we assume the satisfaction of the conditions $y_i^{(m)} \leftarrow y_k^{(r)} \oplus (k, i)$ and $y_i^{(l)} \leftarrow y_j^{(n)} \ominus (i, j)$, then we have that $y_i^{(m)} = (S_i^{(m)}, C_k^{(r)} + c_{ki}, T_k^{(r)} + t_{ki})$ and $y_i^{(l)} = (S_i^{(l)}, C_j^{(n)} + c_{ij}, T_j^{(n)} + t_{ij}, t_i^{(l)}), t_i^{(l)} \equiv t_j^{(n)}$.

The following two situations should be considered:

Case 1.1 Conditions (5.16) are applied.

Case 1.2 Conditions (5.16) are not taken into account.

Case 1.1 In this case, nodes j and k do not belong to the set $\tilde{FS}(i)$ and $\tilde{BS}(i)$, respectively.

First let us consider the node k. In this case, no extensions backward from node i to node k are admitted. This means that no labels associated with node k are generated by the labels of node i. When we apply the $JOIN$ procedure at node k, in the feasible solutions, node i does not appear as a successor of node k, but, instead, the feasible paths with all nodes $h \in \tilde{FS}(k)$ as successors of k are generated. When we apply the $JOIN$ procedure to the node i, the feasible paths with node k as a successor (in the backward direction) of i are generated. This means that, if conditions (5.16) are applied, the feasible paths $\pi_{sd} = \{s = i_1, \ldots, i_{n-1} = k, i_n = h, \ldots, i_l = d\}$ with $l \leq 2$, $k \in \mathcal{N} - sd$ and $n = 2, ..., l - 1$, if they exist, are generated by the $JOIN$ with$h \in FS(k)$, and these paths are all distinct. The same considerations can be made for node j.

Case 1.2 In this second case, for node k, the label $y_k^{(s)} \leftarrow y_i^{(l)} \ominus (k, i)$ is determined and takes the following form, $y_k^{(s)} = (S_k^{(s)}, C_i^{(l)} + c_{ki}, T_i^{(l)} + t_{ki}, t_k^{(s)})$. The $JOIN$ procedure is then applied to node k and, since $y_k^{(r)} \in D^{fw}(k)$ and $y_k^{(s)} \in D^{bw}(k)$, if conditions (5.14)-(5.15) are verified, the union between the two labels is allowed and a path $\pi_{sq}^{(v)} = \pi_{sk}^{(r)} \cup \pi_{kq}^{(s)}$, with the associated label $y_q^{(v)} = (S_q^{(v)}, C_q^{(v)}, T_q^{(v)}) = (S_k^{(r)} \cup S_k^{(s)}, C_k^{(r)} + C_k^{(s)}, T_k^{(r)} + T_k^{(s)} - t_k^{(s)})$, is determined, where:

$$S_d^{(v)}[n_l] = \begin{cases} 0 & if(S_k^{(r)}[n_l] = 0 \wedge S_k^{(s)}[n_l] = 0) \\ 1 & if(S_k^{(r)}[n_l] = 1 \vee S_k^{(s)}[n_l] = 1) \end{cases} n_l = \ \dots, n.$$

We know that $C_k^{(s)} = C_i^{(l)} + c_{ki}$ and $T_k^{(s)} = T^{(l)i} + t_{ki}$, thus $y_q^{(v)} = (S_k^{(r)} \cup S_k^{(s)}, C_k^{(r)} + C_i^{(l)} + c_{ki}, T_k^{(r)} + T_i^{(l)} + t_{ki} - t_i^{(l)})$. In addition, $C_i^{(m)} = C_k^{(r)} + c_{ki}$ and $T_i^{(m)} = T_k^{(r)} + t_{ki}$. On the basis of the previous considerations, it is evident that the label associated with the path $\pi_{sd}^{(v)}$ takes the following form $y_d^{(v)} = (S_k^{(r)} \cup S_k^{(s)}, C_i^{(m)} + C_i^{(l)}, T_i^{(m)} + T_i^{(l)} - t_i^{(l)})$. We know that $S_k^{(r)} = S_i^{(m)} - \{i\}$ and $S_k^{(s)} = S_i^{(l)} \cup \{k\}$; in addition, since $i \in S_i^{(l)}$,we have that $S_i^{(m)} - \{i\} \cup S_i^{(l)} \cup \{k\} \equiv S_i^{(m)} \cup S_i^{(l)}$. Consequently, the condition $\pi_{sd}^{(w)} \equiv \pi_{sd}^{(v)}$ is valid. In other words, the $JOIN$ procedure, which does not consider conditions (5.16), generates equal paths.

(2) Consider a generic arc $(i,j) \in \mathcal{A}$. Let $y_i^{(h)} \in D^{fw}(i), h = 1, \dots, |D^{fw}(i)|$ be the labels associated with node i determined during the forward phase and let $y_j^{(k)} \in Dbw(j), k = 1, \dots, |Dbw(j)|$ be the labels associated with node j determined during the backward phase. Assume that $j \in \tilde{FS}(i)$ and $i \in \tilde{BS}(j)$. At a generic iteration of the algorithm, the label $y_i^{(h)}$ is extracted from L^{fw}. Such a label is extended to node j because $j \in \tilde{FS}(i)$ and the label $y_j^{(l)} \leftarrow y_i^{(h)} \oplus (i,j)$ is added to the set $D^{fw}(j)$. All the labels $y_j^{(k)} \in D^{bw}(j)$ are not extended to node i, because the set $\tilde{BS}(j)$ is adequately updated (i.e., $i \notin \tilde{BS}(j)$). For this reason, no label $y_i^{(n)} \leftarrow y_j^{(k)} \ominus (i,j)$ will be present in $D^{bw}(i)$, thus node j will always belong to the set $\tilde{FS}(i)$ because the condition $\nexists y_i \in D^{bw}(i) : y_i \leftarrow y_j \ominus (i,j)$ is verified at each iteration of the algorithm. Consequently, all the labels y_i generated on the course of the algorithm are extended to j. This means that no potential

feasible solution will be lost. This happens for all arcs $(m, n) \in \mathcal{A}$. Thus we can say that at the end of the algorithm, each node will be associated with all the labels that share the node. Thus, when the $JOIN$ procedure is executed, all feasible paths are generated. This completes the proof. $\square$

The general scheme of the $\mathcal{BDLSM}$, that uses the lower bound rule is outlined in Algorithm 13.

5.4.4 Label selection strategies

Several variants of the label selection methods described in the previous sections have been devised, by considering different label selection policies. It is important to point out that the order in which the labels are selected does not affect the correctness of the proposed methods, but the strategy used to select the label to be processed can influence their efficiency. We have extended the label selection strategies proposed in [76] and [28], to the problem under consideration.

$\mathbf{S}LLF^{\mathbf{C}}$

```
Let $y_{top} = (S_{top},\ C_{top},\ T_{top})$ be the top label of $L$
Let $y_j^{(\xi)} = \left(S_j^{(\xi)},\ C_j^{(\xi)},\ T_j^{(\xi)}\right)$ be a label to be added to $L$
If $C_j^{(\xi)} \leq C_{top}$ Then
    Insert $y_j^{(\xi)}$ at the top of $L$
Else
    insert $y_j^{(\xi)}$ at the bottom of $L$
End If
```

The first label selection policy considered is a modified version of the small lexicographic label first (SLLF, for short) method of [76], in

Algorithm 13 $\mathcal{BDLSM}$ Scheme

```
Step 0 (Initialization phase)
Set:
$D^{fw}(s) = \{y_s^{(1)}\}$, with $S_s^{(1)}[s] = 1, S_s^{(1)}[h] = 0 \;\; \forall h \in \mathcal{N} - \{s\}, T_s^{(1)} = C_s^{(1)} = 0; L^{fw} = \{y_s^{(1)}\}$.
$D^{bw}(d) = \{y_d^{(1)}, y_d^{(2)}, \ldots, y_d^{(b_d - a_d + 1)}\}$, with $S_d^{(k)}[d] = 1, S_d^{(k)}[h] = 0 \;\; \forall h \in \mathcal{N} - \{d\}, T_d^{(k)} = t_d^{(k)}, C_d^{(d)} = 0, t_d^{(k)} = k - 1, k = 1, \ldots, b_d - a_d + 1; L^{bw} = \{y_d^{(1)}, y_d^{(2)}, \ldots, y_d^{(b_d - a_d + 1)}\}$.

Step 1 (Termination check)
if $L^{fw} \cup L^{bw} = \emptyset$ then
    STOP. execute the $JOIN$ procedure in order to determine the optimal path.
end if

Step 2 (Label scan)
Forward Phase
if $L^{fw} \neq \emptyset$ then
    Select a label $y_i^{(m)} = (S_i^{(m)}, C_i^{(m)}, T_i^{(m)})$ of $L^{fw}$ and delate it from $L^{fw}$.
    for all $j \in \tilde{FS}(i)$ do
        if $S_i^{(m)}[j] = 0$ AND $a_j \leq T_i^{(m)} + t_{ij} \leq b_j$ AND $T_i^{(m)} < \frac{T_{max}}{2}$ AND $T_i^{(m)} + t_{ij} + LB_{jd} \leq b_d$ then
            Set: $\bar{T}_j = T_i^{(m)} + t_{ij}; \bar{C}_j = C_i^{(m)} + c_{ij};$
            $\bar{S}_j[w] = S_i^{(m)}[w], \forall w \in \mathcal{N} - \{\{j\} \cup \{l : \bar{T}_j + LB_{jl} > b_l\}\}$.
            $\bar{S}_j[v] = 1, \forall v \in \mathcal{N}, v \neq w$.
            if $(\bar{S}_j, \bar{C}_j, \bar{T}_j)$ is dominated by some label belonging to $D^{fw}(j)$ then
                Go to Step 1.
            else
                Set: $y_j^{(|D^{fw}(j)|+1)} = (\bar{S}_j, \bar{C}_j, \bar{T}_j); D^{fw}(j) = D^{fw}(j) \cup \{y_j^{(|D(j)|+1)}\}; L^{fw} = L^{fw} \cup \{y_j^{(|D^{fw}(j)|+1)}\}$.
                Remove from $D^{fw}(j)$ and $L^{fw}$ all the label that are dominated by $y_j^{(|D^{fw}(j)|+1)}$.
            end if
        end if
    end for
end if
Backrward Phase
if $L^{bw} \neq \emptyset$ then
    Select a label $y_j^{(m)} = (S_j^{(m)}, C_j^{(m)}, T_j^{(m)}, t_j^{(m)})$ of $L^{bw}$ and delate it from $L^{bw}$.
    for all $i \in \tilde{BS}(j)$ do
        if $S_j^{(m)}[i] = 0$ AND $T - b_i \leq T_j^{(m)} + t_{ij} \leq T - a_i$ AND $T_j^{(m)} - t_j^{(m)} < \frac{T_{max}}{2}$ AND $T_j^{(m)} + t_{ij} + LB_{si} \leq b_d$
        then
            Set: $\bar{T}_i = T_j^{(m)} + t_{ij}; \bar{C}_i = C_j^{(m)} + c_{ij}; \bar{t}_i = t_j^{(m)};$
            $\bar{S}_i[w] = S_j^{(m)}[w], \forall w \in \mathcal{N} - \{\{i\} \cup \{l : \bar{T}_i + LB_{li} > T - a_l\}\}$.
            $\bar{S}_i[v] = 1, \forall v \in \mathcal{N}, v \neq w$.
            if $(\bar{S}_i, \bar{C}_i, \bar{T}_i, \bar{t}_i)$ is dominated by some label belonging to $D^{bw}(i)$ then
                Go to Step 1.
            else
                Set: $y_i^{(|D^{bw}(i)|+1)} = (\bar{S}_i, \bar{C}_i, \bar{T}_i, \bar{t}_i); D^{bw}(i) = D^{bw}(i) \cup \{y_i^{(|D^{bw}(i)|+1)}\}; L^{bw} = L^{bw} \cup \{y_i^{(|D^{bw}(i)|+1)}\}$.
                Remove from $D^{bw}(i)$ and $L^{bw}$ all the label that are dominated by $y_i^{(|D^{bw}(i)|+1)}$.
            end if
        end if
    end for
end if
Go to Step 1.
```

which at each iteration the label removed is the top label of L, whereas a new label entering in L is added on the basis of the following rule.

In addition to the rule described above, that considers the value of the cost $C_j^{(\xi)}$ associated to the ξ path $\pi_{sj}^{(\xi)}$ from node s to node j, two other insertion strategies have been defined by considering the other parameters associated to $\pi_{sj}^{(\xi)}$, that is the total time $T_j^{(\xi)}$ and the cost per unit time $R_j^{(\xi)} = \frac{C_j^{(\xi)}}{T_j^{(\xi)}}$.

The related rules are reported in the followings and they are referred to as $SLLF^T$ and $SLLF^R$, respectively.

S$LLF^{\mathbf{T}}$	**S**$LLF^{\mathbf{R}}$
Let $y_{top} = (S_{top},\ \ C_{top},\ \ T_{top})$ be the top label of L	Let $y_{top} = (S_{top},\ \ C_{top},\ \ T_{top})$ be the top label of L
Let $y_j^{(\xi)} = \left(S_j^{(\xi)},\ \ C_j^{(\xi)},\ \ T_j^{(\xi)}\right)$ be a label to be added to L	Let $y_j^{(\xi)} = \left(S_j^{(\xi)},\ \ C_j^{(\xi)},\ \ T_j^{(\xi)}\right)$ be a label to be added to L
If $T_j^{(\xi)} \geq T_{top}$ Then	If $R_j^{(\xi)} \leq R_{top}$ Then
Insert $y_j^{(\xi)}$ at the top of L	Insert $y_j^{(\xi)}$ at the top of L
Else	Else
insert $y_j^{(\xi)}$ at the bottom of L	insert $y_j^{(\xi)}$ at the bottom of L
End If	End If

Label setting like approaches have also been considered. In this case, at each iteration the label existing L is the one with the smallest value of the cost, or the smallest value of the cost per unit time or the greatest value of the time. In what follows, the related versions are referred to as LS^C, LS^R and LS^T, respectively. In order to efficiently extract the label to be processed at each iteration, the list L is maintained partially ordered in a binary heap. Of course, in LS^C each label is placed in the heap on the basis of the value of the cost, in LS^R the insertion in the heap is done on the basis of the value of the cost per unit time, whereas in LS^T the labels are ordered on the basis of the value of the time.

We have also extended the threshold strategy proposed by Chen and Powell ([28]), for the shortest path problem with time windows constraints. In particular, in the original approach, the list of candidate labels L is partitioned in three sub-lists Q_1, Q_2 and Q_3. At any time, the labels in Q_1 are all lexicographically less than those in Q_2 that are lexicographically less than the ones belonging to Q_3 ([28]).

In our case, the Pareto efficiency cannot be applied because of the particular structure of the objective function. Besides the dominance rules are applied only for the labels of the same node and not for the labels belonging to L. Consequently, in order to extend the threshold strategy ([28]) to the problem addressed here, the definition of a pseudo dominance criterion has been introduced.

Given a generic m-th label $y_j^{(m)} = \left(S_j^{(m)}, C_j^{(m)}, T_j^{(m)}\right)$ of node j, a dummy label $\ddot{y}_j^{(m)} = \left(s_j^{(m)},\ R_j^{(m)}\right)$ associated with $y_j^{(m)}$ is defined as follows $\ddot{y}_j^{(m)} = \left(s_j^{(m)},\ f\left(y_j^{(m)}\right)\right)$, where $s_j^{(m)} = -\sum_{k=1}^{n} S_j^{(m)}[k]$ and $R_j^{(m)} = \frac{C_j^{(m)}}{T_j^{(m)}}$. The term $s_j^{(m)}$ represents the number of nodes that belong to the m-th partial path from node s to node j, while $R_j^{(m)}$ is the value of the objective function.

In this case, we want to maximize the function $\sum_{k=1}^{n} S_j^{(m)}[k]$, since we assume that a longer cheaper (in terms of cost per unit time) path is more promising than a path with the same value of the objective function, but with a fewer number of visited nodes.

In order to apply to the dummy labels, introduced above, the well-know Pareto dominance criterion, the value $\sum_{k=1}^{n} S_j^{(m)}[k]$ must be minimized. For this reason, $s_j^{(m)}$ is a negative quantity. Let $\ddot{y}^1$ and $\ddot{y}^2$ be two labels we say that $\ddot{y}^1$ dominates $\ddot{y}^2$ if $\ddot{y}_1^1 = \ddot{y}_1^2$ and $\ddot{y}_2^1 \leq \ddot{y}_2^2$ and at least one inequality is strict.

It is important to point out that the introduced criterion cannot be used to fathom labels, but it has been introduced only to implement the threshold strategy.

At each iteration of the threshold method, the label to be scanned is selected on the basis of the strategy reported in what follows.

Threshold Selection Strategy

```
if $Q_1 \neq \emptyset$ then
   Extract the label $y$ at the bottom of $Q_1$.
   STOP.
end if
if $Q_1 = \emptyset$ then
   if $Q_2 \neq \emptyset$ then
      Transfer all the labels from $Q_2$ to $Q_1$ and set $Q_2 = \emptyset$.
      Extract the label $y$ at the bottom of $Q_1$.
      STOP.
   end if
   if $Q_2 = \emptyset$ then
      if $Q_3 = \emptyset$ then
         the label selection method terminates.
   if $Q_3 \neq \emptyset$ then
      Update the threshold label $y_{thres}$
      Transfer in $Q_1$ all the labels $y$ belonging to $Q_3$ such that the
corresponding labels $\ddot{y}$ are lexicographically less or equal to $y_{thres}$ and
delete them from $Q_3$.
      Extract the label $y$ at the bottom of $Q_1$.
      STOP.
   end if
end if
```

A generic label to be added to the candidate list is inserted to either Q_2 or Q_3 sub-lists on the basis of the following strategy.

Threshold Insertion Strategy

```
**if** $\ddot{y}$ is lexicographically less or equal to $y_{thres}$ **then**
    Insert $y$ at the bottom of $Q_2$
**else**
    Insert $y$ at the bottom of $Q_3$
**end if**
```

The behaviour of the threshold algorithm strongly depends on the strategy used to update the threshold. In our approach, we have followed the updating procedure described in [28].

5.5 Node selection methods

In a node selection approach, at each iteration a node is selected and all the labels associated with the chosen node are evaluated. The main step is represented by the node processing, with the aim to create new labels. In particular, processing a node i involves the examination of all the adjacent nodes j of i (i.e., either $j \in FS(i)$ or $j \in BS(i)$) and the determination of a new set of labels for each node j, by using all the labels belonging to $D(i)$.

It is worth observing that also in this case, it is possible to define different node selection methods, depending on the way the labels associated to the selected node are extended. In particular, a forward (referred to as Forward Node Selection Method, $\mathcal{FNSM}$ for short), a backward (referred to as Backward Node Selection Method, $\mathcal{BNSM}$ for short), and a bi-directional (referred to as Bi-Directional Node Selection Method, $\mathcal{BDNSM}$ for short) versions have been defined.

5.5.1 Froward node selection method

The proposed $\mathcal{FNSM}$ maintains a set of candidate nodes L and, at each iteration, a node i belonging to L is selected. The node i is then processed, that is the set of label $D(i)$ associated with i is used to create a new set of labels for each successors of node i. The algorithm terminates when the list L is found empty, that is to say there are no more nodes left to be processed. The main operations executed by the $\mathcal{FNSM}$, considering the lower bound rule, are reported in Algorithm 14.

Algorithm 14 $\mathcal{FNSM}$ Scheme

```
Step 0 (Initialization phase)
Set: $D(s) = \{y_s^{(1)}\}$, with $S_s^{(1)}[s] = 1, S_s^{(1)}[h] = 0 \ \forall h \in \mathcal{N} - \{s\}, T_s^{(1)} = C_s^{(1)} = 0; L = \{s\}$.

Step 1 (Label selection)
if $L = \emptyset$ then
    STOP.
else
    Select a node $i$ of $L$ and delate it from $L$.
end if

Step 2 (Label scan)
for all $j \in FS(i)$ do
    for all labels $y_i^{(m)} = (S_i^{(m)}, C_i^{(m)}, T_i^{(m)})$ do
        if $S_i^{(m)}[j] = 0$ AND $a_j \leq T_i^{(m)} + t_{ij} \leq b_j$ AND $T_i^{(m)} + t_{ij} + LB_{jd} \leq b_d$ then
            Set: $\bar{T}_j = T_i^{(m)} + t_{ij}; \bar{C}_j = C_i^{(m)} + c_{ij}$;
            $\bar{S}_j[w] = S_i^{(m)}[w], \forall w \in \mathcal{N} - \{\{j\} \cup \{l : \bar{T}_j + LB_{jl} > b_l\}\}$.
            $\bar{S}_j[v] = 1, \forall v \in \mathcal{N}, v \neq w$.
            if $(\bar{S}_j, \bar{C}_j, \bar{T}_j)$ is dominated by some label belonging to $D(j)$ then
                Go to Step 1.
            else
                Set: $y_j^{(|D(j)|+1)} = (\bar{S}_j, \bar{C}_j, \bar{T}_j); D(j) = D(j) \cup \{y_j^{(|D(j)|+1)}\}$.
                Remove from $D(j)$ all the label that are dominated by $y_j^{(|D(j)|+1)}$.
                Add $j$ to $L$ if it does not already belong to it
            end if
        end if
    end for
end for
Go to Step 1.
```

It is evident that the sets $D(j)$, $j \in N$ are updated in such a way that they are undominated sets at all times, that is they contain only efficient labels.

5.5.2 Backward node selection method

In the proposed $\mathcal{BNSM}$, when a node i is selected from the candidate list L, all the labels belonging to $D(i)$ are used to determine a new set of labels for each predecessors of node i. The algorithm terminates when the list L is found empty. The scheme of the $\mathcal{BNSM}$ is reported in Algorithm 15.

Algorithm 15 $\mathcal{BNSM}$ Scheme

```
Step 0 (Initialization phase)
Set: $D(d) = \{y_d^{(1)}, y_d^{(2)}, \dots, y_d^{(b_d - a_d + 1)}\}$, with $S_d^{(k)}[d] = 1, S_d^{(k)}[h] = 0 \quad \forall h \in \mathcal{N} - \{d\}, T_d^{(k)} = t_d^{(k)}, C_d^{(d)} = 0, t_d^{(k)} = k - 1, k = 1, \dots, b_d - a_d + 1; L = \{d\}$.

Step 1 (Label selection)
if $L = \emptyset$ then
    STOP.
else
    Select a node $j$ of $L$ and delate it from $L$.
end if

Step 2 (Label scan)
for all $i \in BS(j)$ do
    for all labels $y_j^{(m)} = (S_j^{(m)}, C_j^{(m)}, T_j^{(m)}, t_j^{(m)}) \in D(j)$ do
        if $S_j^{(m)}[i] = 0$ AND $T - b_i \leq T_j^{(m)} + t_{ij} \leq T - a_i$ AND $T_j^{(m)} + t_{ij} + LB_{si} \leq b_d$ then
            Set: $\bar{T}_i = T_j^{(m)} + t_{ij}; \bar{C}_i = C_j^{(m)} + c_{ij}; \bar{t}_i = t_j^{(m)}$;
            $\bar{S}_i[w] = S_j^{(m)}[w], \forall w \in \mathcal{N} - \{\{i\} \cup \{l : \bar{T}_i + LB_{li} > T - a_l\}\}$.
            $\bar{S}_i[v] = 1, \forall v \in \mathcal{N}, v \neq w$.
            if $(\bar{S}_i, \bar{C}_i, \bar{T}_i, \bar{t}_i)$ is dominated by some label belonging to $D(i)$ then
                Go to Step 1.
            else
                Set: $y_i^{(|D(i)|+1)} = (\bar{S}_i, \bar{C}_i, \bar{T}_i, \bar{t}_i); D(i) = D(i) \cup \{y_i^{(|D(i)|+1)}\}$.
                Remove from $D(i)$ all the label that are dominated by $y_i^{(|D(i)|+1)}$.
                Add $i$ to $L$ if it does not already belong to it.
            end if
        end if
    end for
end for
Go to Step 1.
```

5.5.3 Bi-directional node selection method

Similarly to the label selection counterpart, the proposed $\mathcal{BDNSM}$ to address the $\mathcal{LFESPPTW}$ can be viewed as characterized by the execution of three main steps: the forward, the backward and the join steps.

The method maintains two list of candidate nodes, that is L^{fw} and L^{bw}, in which the nodes, for which at least a new label has been determined during either the forward or the backward phase, respectively, are stored.

For each node $i \in N$ the labels set is viewed as the union of two sets, that is $D(i) = D^{fw}(i) \cup D^{bw}(i)$, where $D^{fw}(i)$ contains the labels generated during the forward phase (i.e., forward labels) and $D^{bw}(i)$ those determined during the backward step (i.e., backward labels). The algorithm terminates when the set $L^{fw} \cup L^{bw}$ is found empty. At this point, the optimal path is obtained by executing the $JOIN$ procedure, described in Section 5.4.3. It is worth observing that all the bounding rules introduced in the case of the $\mathcal{BDLSM}$ apply to the approach presented in this section. The steps of the proposed $\mathcal{BDNSM}$ are reported in Algorithm 16.

The sets $\widetilde{FS}(i)$ and $\widetilde{BS}(j)$ have been defined in Section 5.4.3.

5.5.4 Node selection strategies

The efficiency of the node selection methods described in the previous section depends on the strategy used to select, at each iteration, the node belonging to L [or L^{bw} and L^{fw}], to be processed.

It is worth observing that very efficient versions can be obtained, when acyclic networks are considered. Indeed, in this case it is possible to scan the nodes by following their topological order. In what follows, the related approach is denoted as Topological Order Strategy ($\mathcal{TOS}$, for short).

For general networks, three different node selection strategies have been defined.

The first one has been obtained by extending the small average label selection strategy ($\mathcal{SALS}$, for short) developed for the multicriteria shortest path problem in [76] to the $\mathcal{LFESPPTW}$.

Algorithm 16 $\mathcal{BDNSM}$ Scheme

```
Step 0 (Initialization phase)
Set:
$D^{fw}(s) = \{y_s^{(1)}\}$, with $S_s^{(1)}[s] = 1, S_s^{(1)}[h] = 0 \;\; \forall h \in \mathcal{N} - \{s\}, T_s^{(1)} = C_s^{(1)} = 0; L^{fw} = \{s\}$.
$D^{bw}(d) = \{y_d^{(1)}, y_d^{(2)}, \dots, y_d^{(b_d - a_d + 1)}\}$, with $S_d^{(k)}[d] = 1, S_d^{(k)}[h] = 0 \;\; \forall h \in \mathcal{N} - \{d\}, T_d^{(k)} = t_d^{(k)}, C_d^{(d)} = 0, t_d^{(k)} = k - 1, k = 1, \dots, b_d - a_d + 1; L^{bw} = \{d\}$.

Step 1 (Termination check)
if $L^{fw} \cup L^{bw} = \emptyset$ then
    STOP. execute the $JOIN$ procedure in order to determine the optimal path.
end if

Step 2 (Label scan)
Forward Phase
if $L^{fw} \neq \emptyset$ then
    Select a node i belongingto $L^{fw}$ and delate it from $L^{fw}$.
    for all $j \in \tilde{FS}(i)$ do
        for all $y_i^{(m)} = (S_i^{(m)}, C_i^{(m)}, T_i^{(m)}) \in D^{fw}(i)$ do
            if $S_i^{(m)}[j] = 0$ AND $a_j \leq T_i^{(m)} + t_{ij} \leq b_j$ AND $T_i^{(m)} < \frac{T_{max}}{2}$ AND $T_i^{(m)} + t_{ij} + LB_{jd} \leq b_d$ then
                Set: $\bar{T}_j = T_i^{(m)} + t_{ij}; \bar{C}_j = C_i^{(m)} + c_{ij}$;
                $\bar{S}_j[w] = S_i^{(m)}[w], \forall w \in \mathcal{N} - \{\{j\} \cup \{l : \bar{T}_j + LB_{jl} > b_l\}\}$.
                $\bar{S}_j[v] = 1, \forall v \in \mathcal{N}, v \neq w$.
                if $(\bar{S}_j, \bar{C}_j, \bar{T}_j)$ is dominated by some label belonging to $D^{fw}(j)$ then
                    Go to Step 1.
                else
                    Set: $y_j^{(|D^{fw}(j)|+1)} = (\bar{S}_j, \bar{C}_j, \bar{T}_j); D^{fw}(j) = D^{fw}(j) \cup \{y_j^{(|D(j)|+1)}\}$.
                    Remove from $D^{fw}(j)$ all the label that are dominated by $y_j^{(|D^{fw}(j)|+1)}$.
                    Add $j$ to $L^{fw}$ if it does not already belong to it.
                end if
            end if
        end for
    end for
end if
Backrward Phase
if $L^{bw} \neq \emptyset$ then
    Select and delate a node $j$ belonging to $L^{bw}$.
    for all $i \in \tilde{BS}(j)$ do
        for all $y_j^{(m)} = (S_j^{(m)}, C_j^{(m)}, T_j^{(m)}, t_j^{(m)}) \in D^{bw}(j)$ do
            if $S_j^{(m)}[i] = 0$ AND $T - b_i \leq T_j^{(m)} + t_{ij} \leq T - a_i$ AND $T_j^{(m)} - t_j^{(m)} < \frac{T_{max}}{2}$ AND $T_j^{(m)} + t_{ij} + LB_{si} \leq b_d$
            then
                Set: $\bar{T}_i = T_j^{(m)} + t_{ij}; \bar{C}_i = C_j^{(m)} + c_{ij}; \bar{t}_i = t_j^{(m)}$;
                $\bar{S}_i[w] = S_j^{(m)}[w], \forall w \in \mathcal{N} - \{\{i\} \cup \{l : \bar{T}_i + LB_{li} > T - a_l\}\}$.
                $\bar{S}_i[v] = 1, \forall v \in \mathcal{N}, v \neq w$.
                if $(\bar{S}_i, \bar{C}_i, \bar{T}_i, \bar{t}_i)$ is dominated by some label belonging to $D^{bw}(i)$ then
                    Go to Step 1.
                else
                    Set: $y_i^{(|D^{bw}(i)|+1)} = (\bar{S}_i, \bar{C}_i, \bar{T}_i, \bar{t}_i); D^{bw}(i) = D^{bw}(i) \cup \{y_i^{(|D^{bw}(i)|+1)}\}$.
                    Remove from $D^{bw}(i)$ all the label that are dominated by $y_i^{(|D^{bw}(i)|+1)}$.
                    Add $i$ to $L^{bw}$ if it does not already belong to it.
                end if
            end if
        end for
    end for
end if
Go to Step 1.
```

The main idea is to select, at each iteration, the node i belonging to L for which the best average value label σ_i in $D(i)$ is obtained. Since each label is characterized by the time and the cost and σ_i is defined as the sum of the values of the labels associated to i divided by the cardinality $|D(i)|$ of $D(i)$, σ_i can be determined by considering the time (the corresponding value is denoted by $\sigma{}_i^T$) the cost (denoted as $\sigma{}_i^c$) or the cost per unit time (indicated as $\sigma{}_i^R$). In the sequel, the corresponding strategies are referred to as $\mathcal{SALS}^T$, $\mathcal{SALS}^C$ and $\mathcal{SALS}^R$ respectively.

The generic scheme of the $\mathcal{SALS}$ rules can be summarized as follows.

$\mathcal{SALS}$

Step.1 For all nodes $j \in L$ determine

$$\sigma_j^T = \frac{\sum_{k=1}^{|D(j)|} T_j^{(k)}}{|D(j)|} \left[or\ \sigma_j^C = \frac{\sum_{k=1}^{|D(j)|} C_j^{(k)}}{|D(j)|} \right] \left[or\ \sigma_j^R = \frac{\sum_{k=1}^{|D(j)|} R_j^{(k)}}{|D(j)|} \right]$$

Step.2 Remove from L a node i such that the following condition is satisfied:

$$i = \arg\max \left\{ \sigma_j^T \ : j \in L \right\} \quad \left[or\ i = \arg\min \left\{ \sigma_j^C \ : j \in L \right\} \ \right]$$
$$\left[or\ i = \arg\min \left\{ \sigma_j^R \ : j \in L \right\} \ \right]$$

In order to extract the node i in an efficient way, the list L is maintained partially ordered in a binary heap. Indeed, each node i is placed in the heap on the basis of the value of $\sigma{}_i^T$ ($\sigma{}_i^c$ or $\sigma{}_i^R$).

In addition to the selection rules described above, another policy has been introduced where the node i is selected on the basis of the best value of the labels belonging to $D(i)$. The corresponding rule is defined best label selection strategy ($\mathcal{BLS}$, for short) and it can be defined by considering the time, the cost or the cost per unit time.

The related methods, which will be referred to herein as $\mathcal{BLS}^T$, $\mathcal{BLS}^C$ and $\mathcal{BLS}^R$, respectively, are reported in what follows.

$\mathcal{BLS}$

Step.1 For all nodes $j \in L$ determine

$\sigma_j^T = \max_{k=1,\ldots,|D(j)|} T_j^{(k)}$, $\left[or\ \sigma_j^T = \min_{k=1,\ldots,|D(j)|} C_j^{(k)} \right]$,

$\left[or\ \sigma_j^R = \min_{k=1,\ldots,|D(j)|} R_j^{(k)} \right]$

Step.2 Remove from L a node i such that the following condition is satisfied:

$i = \arg\max \left\{ \sigma_j^T \ : j \in L \right\}$ $\left[or\ i = \arg\min \left\{ \sigma_j^C \ : j \in L \right\} \right]$

$\left[or\ i = \arg\min \left\{ \sigma_j^R \ : j \in L \right\} \right]$

We have considered also an extension of the small average label first strategy ($\mathcal{SALF}$, for short) proposed in [76]. More specifically, the value $\sigma\ _i^T$ ($\sigma\ _i^c$ or $\sigma\ _i^R$). associated with a node i is used to insert it into list L, whereas, at each iteration, the removed node is always the top node of L. The corresponding strategies will be subsequently referred to as $\mathcal{SALF}^T$, $\mathcal{SALF}^C$ and $\mathcal{SALF}^R$, respectively.

In these approaches, a node j is added to L on the basis of the following insertion rule.

$\mathcal{SALF}$

```
Let $j$ be a node to be added to $L$
Let $i$ be the top node of $L$
If $\sigma_j^T \geq \sigma_i^T$ $[\ or\ \sigma_j^C \leq \sigma_i^C]$ $[\ or\ \sigma_j^R \leq \sigma_i^R]$ Then
Insert $j$ at the top of $L$
Else
Insert $j$ at the bottom of $L$
```

End If

5.6 Computational experiments

The main goal of this section is to analyze the numerical behaviour of the proposed multi-dimensional labelling approaches to address the $\mathcal{LFESPPTW}$ when different label and node selection strategies are used. The influence of the proposed dominance rules on the computational cost is also investigated.

The considered solution approaches have been implemented in java language and have been tested by using an Intel(R) Core(TM)2QUAD CPU Q6600 PC, 1.39 GHz, RAM 4 GB, under the Microsoft Vista operating system. All the label and node selection methods described in the previous sections have been tested. Indeed, three different versions of each method have been considered: a naive version (i.e., 1^{st}), in which no dominance checks are applied and two enhanced versions in which, respectively, only the first dominance check (i.e., 2^{nd}) and the second dominance rule is applied (i.e., 3^{rd}), respectively. In what follows, we shall refer to each implemented algorithm as *approach.selection strategy.version*. Thus the algorithm $\mathcal{FLSM}.\mathcal{SLLF}.1^{st}$ denotes the naive version of the $\mathcal{FLSM}$, that uses the $\mathcal{SLLF}$ selection strategy.

As far as the choice of the test problems is concerned, we have considered three classes of instances of varying size and density (defined as the ratio between the number of arcs and the number of nodes).

In particular, fully random (i.e., SpRand networks), acyclic random (i.e., SpAcyc networks) and complete graphs (i.e., Solomon's networks) have been considered. For the first two sets of test problems, the arc times and costs are chosen according to a uniform distribution from the range $[0,\ 100]$.

The time windows have been determined as follows. For each node

$i \in N$, the centre of the associated time window is set equal to the minimum time T required to reach node i starting from node s. A constant AVG is used to define the interval $[a_i; b_i]$ as $\left[T - \frac{AVG}{2}; T + \frac{AVG}{2}\right]$.

The SpRand and the SpAcyc networks have been generated by using a modified version of the public domain SPRAND and SPACYC generators ([29]). As these codes generate, for each arc (i, j) only the parameter c_{ij}, representing the cost, the modification entails using the SPRAND and SPACYC pseudo-random number generator to also generate the time parameters t_{ij}, associated to each arc, from a specified range. The characteristics of the test problems are reported in Table 5.1, where for each network, the number of nodes, the number of arcs and the density value are highlighted.

The SpRand networks $(R_1 - R_{15})$ are generated by first creating a Hamiltonian cycle and then adding arcs with distinct random end points. In the SpAcyc networks $(AR_1 - AR_{15})$, the nodes are numbered from 1 to $|N|$, and there is a path of arcs $(i, i+1)$, $1 \leq i \leq |N|$. Additional arcs are generated by picking two distinct nodes at random and creating an arc from the lower to the higher numbered node.

It is worth observing that when acyclic networks are considered, the dummy resource associated to each node is not taken into account, since all the paths do not contain repeated nodes.

The computational experiments have been carried out by considering, for each test problem, four different values for the parameter AVG, that is 40, 60, 80 and 100. In the following, we shall refer to the test problem by using the notation *test.time windows structure*. Thus, problem $R_1.I$ is problem R_1 for which the time windows have been defined by setting AVG=40. In all the experiments, we set the origin node $s = 1$ and the destinations node $d = |N|$.

The third set of test problems have been derived from the well-known Solomon's data-set of the vehicle routing problem with time windows, in which the vehicle capacity is not taken into account.

Test	Nodes	Arcs	Density
$AR_1 - R_1$	2500	30000	12
$AR_2 - R_2$	2000	30000	15
$AR_3 - R_3$	1500	30000	20
$AR_4 - R_4$	1000	30000	30
$AR_5 - R_5$	500	30000	60
$AR_6 - R_6$	2500	50000	20
$AR_7 - R_7$	2000	50000	25
$AR_8 - R_8$	1500	50000	33
$AR_9 - R_9$	1000	50000	50
$AR_{10} - R_{10}$	500	50000	100
$AR_{11} - R_{11}$	2500	60000	24
$AR_{12} - R_{12}$	2000	60000	30
$AR_{13} - R_{13}$	1500	60000	40
$AR_{14} - R_{14}$	1000	60000	60
$AR_{15} - R_{15}$	500	60000	120

Table 5.1: Characteristics of the SpRand and SpAcyc Networks.

These instances and a description of their characteristics are available at *http://web.cba.neu.edu/˜msolomon/problems.htm*. The problems with 100 nodes have been considered. The times for travelling from one node to the others are set equal to the corresponding distance, while the costs are generated randomly according to a uniform distribution from the range [0, 100]. This data-set is divided into clustered (i.e., $c101 - c109$), random (i.e., $r101 - r112$) and random-clustered (i.e., $rc101 - rc108$) categories, according to the displacement of the nodes. Instances belonging to the same data-set have the nodes located in the same way, the instances differ only for the width of the time windows.

5.6.1 Numerical results

The performances of the proposed methods have been evaluated by considering the CPU times (in milliseconds), the number of discarded labels, when the dominance criteria are applied and the number of iterations, averaged over thirty different runs. It is worth observing that in the case of label selection methods, an iteration corresponds to the execution of the scanning of a label, whereas for the node selection methods an iteration corresponds to the processing of a node. For the bi-directional counterparts, an iteration corresponds to the execution of both the forward and the backward phases. A detailed accounting of the experimental results is reported in [54]. They underline that it is not a trivial task to choose the parameter, among time, cost and cost per unit time, for selecting the node/label to be processed that always allows the achievement of the best performance. Thus, in what follows we consider the best results, in terms of computational cost, obtained by applying the implemented algorithms. The related results are given in the Appendix A. The main aim of the experimental phase is to evaluate the impact of the proposed dominance rules as well as that the node/label selection strategies on the computational cost required by the proposed approaches to solve the $\mathcal{LFESPPTW}$. In order to individuate the best performing node and label-selection strategies, computational experiments have been carried out by considering only the first two sets of test problems (i.e., SpRand and SpAcyc networks). The versions, which result the most efficient, have been then applied to the Solomon's networks. The related results are presented in what follows.

Numerical results on SpRand and SpAcyc networks

In this section, we present the computational results obtained by applying the proposed solution approaches to the SpRand and the SpAcyc networks.

Dominance rules evaluation. A first set of experiments has been carried out with the goal of evaluating how the use of the proposed dominance rules may affect the computational cost. It is worth observing that, for the SpRand networks it is possible to consider only the first dominance criterion, whereas for the SpAcyc networks the proposed second rule, that includes the first one as special case, has been applied.

The results obtained when solving the SpRand networks are reported in Tables A.1 and A.2 of the Appendix A; those related to the Acyclic networks are given instead in Tables A.3-A.6 of the Appendix A, where for each AVG value, we report the percentage improvement, in the execution time and in the number of iterations, observed by applying the dominance rules as compared to the naive version of the proposed approaches. The number of discarded labels is also given. It is important to point out that some of the proposed approaches run out of memory when solving $R_{15}.IV$. Consequently, the values reported in Tables A.1 and A.2 of the Appendix A have been determined without taking into account the results obtained on this test problem.

The computational results collected show that the introduction of the dominance criteria allows the cut-off of unpromising labels and thus a reduction in the number of iterations and in the computational cost is observed.

More specifically, in the case of the SpRand networks, the results related to the $\mathcal{FNSM}s$ (see Table A.1) underline an average percentage reduction in the number of iterations of 24.72%, 32.60%, 33.51% and 27.18% for AVG= 40, 60, 80 and 100, respectively. A similar trend has been observed in the reduction of the computational cost (i.e., a percentage reduction of 64.03%, 57.87%, 50.34% and 46.21%, for AVG= 40, 60, 80 and 100, respectively). In the case of the $\mathcal{BDNSM}s$ (see Table A.1), for which even the number of labels generated by the naive approach is lower than those determined by the forward counterpart, the advantage of applying the first dominance rule is less evident. In addition, the average number of discarded labels is equal to 61000.74 and 23173.28 for the $\mathcal{FNSM}s$ and the $\mathcal{BDNSM}s$, respectively; whereas the average percentage reduction in the number of iterations [and in the computational time] is of 29.50 % and 19.89% [54.61% and 48.04%], for the $\mathcal{FNSM}s$ and the bidirectional counterparts, respectively.

As the results of Table A.2 underline, a trend similar to that obtained for the node selection methods is also observed for the label selection approaches. In particular, the introduction of the dominance rule in the $\mathcal{FNSM}s$ allows an average percentage reduction of the 91.77% and 91.01% in the number of iterations and in the computational cost, respectively, whereas in the case of the bidirectional counterparts, the average percentage reduction in the iterations and in the computational overhead is more modest and it is equal to 56.00% and 55.82%, respectively. In addition, the number of discarded labels is equal to 67472.07 and 8571.69, for the forward and the bi-directional versions, respectively.

Tables A.3 and A.4 display the computational results obtained by the node selection methods (forward and bi-directional) when solving the SpAcyc networks. As expected, the gain of using the second dominance criterion is greater than the one observed when the first rule is applied.

Indeed, for the $\mathcal{FNSM}s$, when the first rule is adopted, the average percentage reduction in the number of iterations and in the execution time is equal to 25.26% and 69.70%, respectively. The gain increases to 27.52% and 71.01% when the second rule is applied. A similar trend is observed for the bi-directional counterparts (see Table A.4).

Tables A.5 and A.6 provide statistics on the influence of dominance rules in the solution process of the label selection methods when solving the SpAcyc networks. These results underline a trend similar to that obtained with the node selection methods.

Indeed, for the forward versions (see Table A.5), the average percentage reduction in the number of iterations [in the computational cost] is 91.80 % and 92.62% [92.31% and 93.00%] when the first dominance rule and the second dominance criterion are applied, respectively.

On the other hand, as the results of Table A.6 show, the gain obtained with the bi-directional version of the label selection methods is less impressive. Indeed, the application of the first criterion allows, on average, a percentage reduction in the number of iterations and in the execution time of the 58.48% and 58.63%, respectively. The average percentage improvement in the number of iterations and in the computational cost increases, respectively, to the 61.41% and 61.39% when the second dominance rule is applied.

Node selection methods results. The results collected on the node selection methods are summarized in Tables A.7 and A.8 of Appendix A. For each instance and for each method, we give the computational cost (in milliseconds) and the number of iterations that were obtained by applying the best performing version of the related approach, when solving the SpRand and the SpAcyc networks, respectively.

The computational results of Table A.7 show that, in the case of the SpRand networks all the forward versions fail to solve the test $R_{15}.IV$. On the other hand, the bi-directional counterpart allows the determination of the optimal solution for all the considered networks and BDNSM.SALS.2nd shows the best behaviour when solving $R_{15}.IV$. In order to compare the forward and the bi-directional versions, the results obtained on $R_{15}.IV$ have not been considered. Table A.7 underlines that the bi-directional algorithms with bounding outperform the forward counterparts. Indeed, the forward versions are on average 1015.53 times slower than the bi-directional versions. This behaviour can be explained by taking into account the reduction in the number of iterations. In particular, the number of iterations required by the $\mathcal{FNSM}s$ is on average 8.96 times greater than that of the bi-directional ones.

As far as the numerical comparison among the $\mathcal{BDNSM}s$ is concerned, a significant observation is that $\mathcal{SALS}$ is on average the fastest followed by $\mathcal{SALF}$ and $\mathcal{BLS}$, when solving the SpRand networks. More specifically, $\mathcal{BDNSM}.\mathcal{SALS}.2^{nd}$ is 2.09 and 3.42 times faster than $\mathcal{BDNSM}.\mathcal{SALF}.2^{nd}$ and $\mathcal{BDNSM}.\mathcal{BLS}.2^{nd}$, respectively.

The computational results collected on the SpAcyc networks are reported in Table A.8 of Appendix A. By comparing the numerical results of the forward versions with those related to the bi-directional counterparts, a trend similar to that obtained for the SpRand networks is observed. Indeed, without considering the $\mathcal{TOS}$ method for which a bi-directional counterpart has not been defined, the bi-directional approaches dominate the forward ones. In particular, the formers are on average 1.22 times faster than the latter and require 2.10 times less iterations.

Table A.8 underlines also that $\mathcal{FNSM}.\mathcal{TOS}.3^{rd}$ is on average the fastest among the node selection methods, requiring on average 1500.00 iterations and a computational cost of 226.74 milliseconds. The second best node selection approach is $\mathcal{BDNSM}.\mathcal{SALF}.3^{rd}$ followed by $\mathcal{BDNSM}.\mathcal{SALS}.3^{rd}$ and then $\mathcal{BDNSM}.\mathcal{BLS}.3^{rd}$. In particular, $\mathcal{BDNSM}.\mathcal{SALF}.3^{rd}$ is on average 1.46 and 2.06 times faster than $\mathcal{BDNSM}.\mathcal{SALS}.3^{rd}$ and $\mathcal{BDNSM}.\mathcal{BLS}.3^{rd}$, respectively. This behaviour can be explained by taking into account the number of iterations performed by the methods (see Table A.8).

Label selection methods results. Tables A.9 and A.10 of Appendix A provide statistics on the solution process of the forward and bi-directional label selection approaches, when solving the SpRand and SpAcyc networks, respectively. Similarly to that done for the node selection methods, for each test problem and for each method, we report the computational cost and the number of iterations of the best performing version of the related approach.

The results collected clearly underline that the forward versions remarkably outperform the bi-directional counterparts with the sole exception of $\mathcal{THR}$, for which the trend is inverted. Indeed, $\mathcal{BDLSM}.\mathcal{THR}.2^{nd}$ is 30.59 times faster than the $\mathcal{FLSM}.\mathcal{THR}.2^{nd}$. It is worth noting that the $\mathcal{THR}$ approaches fail to solve $R_{15}.IV$. The average execution time of the $\mathcal{FLSM}s$ when solving the SpRand networks, without considering the $\mathcal{THR}$ strategy, is about 963.72 milliseconds, whereas the average computational cost of the bi-directional versions is equal to 7382.26 milliseconds (see Table A.9). This behaviour can be explained by comparing the number of iterations executed by the two versions of each algorithm: the average number of iterations of the bi-directional versions is about 1.63 times greater than that of the forward counterparts.

A similar trend is observed for the SpAcyc networks, for which the $\mathcal{BDLSM}s$ is 10.23 times slower than the $\mathcal{FLSM}s$ with a number of iteration 1.10 greater than those required by the $\mathcal{FLSM}s$. As far as the numerical comparison among the label selection methods is concerned the computational results of Tables A.9 underline that $\mathcal{FIFO}$, $\mathcal{SLLS}$ and $\mathcal{LS}$ show comparable performance when solving SpRand networks. Indeed, in all the considered test problems $\mathcal{LS}$ behaves the best, followed by $\mathcal{SLLF}$ and $\mathcal{FIFO}$. The $\mathcal{THR}$ is very inefficient to solve the problem on random networks. However, the improvement is very slight overall as reflected by the insignificant change in the number of iterations. Also for SpAcyc networks, the computational results of Tables A.10 underline that they show comparable performance. Indeed, in all the considered test problems $\mathcal{THR}$ is the most efficient selection strategy, followed by $\mathcal{LS}$, $\mathcal{SLLF}$ and $\mathcal{FIFO}$.

The performance of the proposed label selection methods has been compared with the approach proposed by J. Roan and C. Lee in [122], referred to as $\mathcal{RLA}$, that represents the only approach proposed in the scientific literature to address the problem under study. The numerical results obtained when testing $\mathcal{RLA}$, on the considered problems, are reported in Table A.11, in which the execution time (in milliseconds) and the number of iterations are presented.

The results show that, the proposed label correcting methods always outperform the $\mathcal{RLA}$ and they are substantially faster than the $\mathcal{RLA}$. Indeed, the best performing label selection method is on average 173.12 and 58.86 times faster than $\mathcal{RLA}$ in solving the SpAcyc and SpRand networks, respectively.

Comparison and discassion. The main aim of this section is to make a comparison among the proposed node selection and the label selection methods, when solving the first two sets of considered test problems.

The computational results obtained by testing the $\mathcal{LFESPPTW}$ on random networks (see Tables A.7 and A.9 of the Appendix A), indicate that the node selection approaches are on average 3.31 times faster than the label selection methods.

On the acyclic networks, the $\mathcal{BDLSM}.\mathcal{FIFO}.3^{rd}$ shows the worst performance. In addition, the computational results demonstrate clearly that $\mathcal{FNSM}.\mathcal{TOS}.3^{rd}$ is the fastest approach. This conclusion agrees with the numerical experience for the classical shortest path problem in acyclic graphs, for which the topological node selection approach is always superior.

As shown in Tables A.10 and A.8 of Appendix A, the speedup factor seems to increase with the window width. Indeed, $\mathcal{FNSM}.\mathcal{TOS}.3^{rd}$ is 5.26, 12.81, 24.45 and 45.27 times faster than $\mathcal{BDLSM}.\mathcal{FIFO}.3^{rd}$ for AVG=40, 60, 80 and 100 respectively. This behaviour can be explained by observing that larger time windows determine a greater number of labels associate with each node. Thus in a node selection method, we have a larger cost for iteration. For this reason, a node selection strategy could show better performance than a label selection approach only if a small number of iterations is executed; this condition is surely satisfied by the $\mathcal{TOS}$ approach, in which each node is processed exactly once and this issue also explains the reason for which the remaining node selection methods show a worse performance compared to the label selection approaches.

We remark that the bidirectional versions outperform the forward counterparts only when considering the node selection strategy. Indeed, the forward versions are the most efficient considering the label selection strategy. This behaviour is seen for both SpRand and SpAcyc networks.

The $\mathcal{FLSM}s$ are on average 7.66 and 10.23 times faster than the $\mathcal{BDLSM}s$, when solving the SpRand and SpAcyc networks, respectively. On the other hand, the $\mathcal{FNSM}s$ are on average 259.51 and 1.22 times slower than the $\mathcal{BDNSM}s$, when solving the SpRand and SpAcyc networks, respectively. The results explain that the bidirectional approach is efficient only with a node selection strategy. When a label selection strategy is used, the fathoming rule introduced by the bi-directional approach does not suffice to guarantee an efficient behaviour.

Computational results on Solomon's instances

The best performing versions of the proposed label and node selection approaches have been applied to the Solomon's networks. In particular, the computational experiments have been carried out by considering the $\mathcal{LS}$ and the $\mathcal{SALS}$ forward and bi-directional strategies (i.e., $\mathcal{FLSM}.\mathcal{LS}.2^{nd}$, $\mathcal{BDLSM}.\mathcal{LS}.2^{nd}$, $\mathcal{FNSM}.\mathcal{SALS}.2^{nd}$ and $\mathcal{BDNSM}.\mathcal{SALS}.2^{nd}$) that, on the basis of the numerical results presented in the previous sections, turneded out to be the most efficient approaches to solve the $\mathcal{LFESPPTW}$. The related results are reported in Tables A.12 and A.13 of Appendix A.

The computational results collected on the label selection methods underline that in three cases (i.e., c103, c104 and c109) $\mathcal{FLSM}.\mathcal{LS}.2^{nd}$ runs out of memory, whereas $\mathcal{BDLSM}.\mathcal{LS}.2^{nd}$ is not able to solve only the problem c104. On the remaining instances, a behaviour similar to the one obtained for the SpRand networks is observed. Indeed, $\mathcal{FLSM}.\mathcal{LS}.2^{nd}$ outperforms the bidirectional counterpart, even though the gain obtained is limited (i.e., $\mathcal{FLSM}.\mathcal{LS}.2^{nd}$ is 1.20times faster than $\mathcal{BDLSM}.\mathcal{LS}.2^{nd}$).

As far as the performance of the node selection methods is concerned, Table A.13 underlines that BDNSM.SALS.2nd solves all the considered complete networks, whereas the forward counterpart fails to solve the problems c103 and c104. On the remaining networks, $\mathcal{BDNSM}.\mathcal{SALS}.2^{nd}$ outperforms $\mathcal{FNSM}.\mathcal{SALS}.2^{nd}$. In particular, the former is on average 3.19 times faster and requires 1.20 times less iterations. This behaviour is consistent with the one observed by Righini et al. [119], where a node selection approach is proposed to solve the elementary shortest path problem with time windows and vehicle capacity constraints.

Tables A.12 and A.13 indicate that also on this set of test problems the node selection approaches outperform the label selection methods. In particular, on the Solomon's networks, the best performing node selection method (i.e., $\mathcal{BDNSM}.\mathcal{SALS}.2^{nd}$) turns out to be 5.82 times faster than the most efficient label selection approach (i.e., $\mathcal{FLSM}.\mathcal{LS}.2^{nd}$).

5.7 Conclusions

In this paper we have focused on the solution of the elementary linear fractional shortest path problem with time windows. We have presented multi-dimensional labelling approaches to address the problem under consideration, defined by using different label selection and node selection strategies, on the basis of which the most promising node/label is selected at each iteration. In order to define efficient solution approaches, a bi-directional search strategy and some dominance rules, well tailored to the problem at hand, have also been exploited.

In order to assess the behaviour of the proposed approaches, extensive computational experiments have been carried out on a set of randomly generated networks, with varying size and density. The label selection methods have been compared with the state of art approach to address the problem under study, i.e. the label correcting method in which the candidate list of labels is accessed by a FIFO policy proposed by Roan and Lee in 2003 [122].

The experimental results have shown that: 1) the introduction of the proposed dominance rules allow the cut-off of unpromising paths and thus an improvement in the execution time is obtained; 2) the bi-directional versions of the proposed approaches are faster than the forward counterparts only with node selection strategy; 3) the proposed label selection approaches turn out to be better in terms of execution time than the FIFO algorithm of Roan and Lee; 4) for random networks, the node selection methods are generally faster than the label selection approaches; 5) for acyclic networks, the topological order node selection approach is the best performing method; 6) for Solomon's networks the node selection strategy is more efficient than the label selection approach and the bi-directional node selection method outperforms the forward counterpart.

In conclusion, it is worth observing that in this paper we have addressed the linear fractional shortest path problem with hard time windows constraints. It could be interesting to investigate the case in which soft time window constraints are associated to the nodes. This represents the subject of current investigation.

Achnowledgements

The authors wish to acknowledge the contribution of Dr Simona Benigno in the development of an earlier version of the paper. They also wish to thank the anonymous referees for their constructive suggestions and comments.

Appendix A - Computational Results

		$\mathcal{FNSM}$			$\mathcal{BDNSM}$		
		$\mathcal{SALS}$	$\mathcal{BLS}$	$\mathcal{SALF}$	$\mathcal{SALS}$	$\mathcal{BLS}$	$\mathcal{SALF}$
AVG=40	Time	55.69%	89.04%	47.37%	31.38%	23.31%	24.18%
	# Dom	12181.41	12168.1	9494.04	2614.7	2596.59	2864.27
	# Iter.	15.22%	56.30%	2.65%	16.80%	15.91%	0.75%
AVG=60	Time	46.17%	84.32%	43.11%	35.37%	41.73%	38.94%
	# Dom	36560.6	37072.51	28832.67	10016.3	10385.64	9964.54
	# Iter.	37.25%	55.96%	4.59%	11.89%	31.96%	15.09%
AVG=80	Time	38.89%	78.08%	34.04%	70.41%	75.40%	77.41%
	# Dom	84012.76	86734.59	66412.98	24472.45	27625.55	24174.09
	# Iter.	37.35%	55.12%	8.05%	13.09%	39.91%	35.35%
AVG=100	Time	33.48%	75.77%	29.38%	43.01%	56.98%	58.33%
	# Dom	128399.82	129571.69	100561.69	53751.02	57635.87	51978.36
	# Iter.	29.30%	47.93%	4.32%	48.27%	1.10%	8.52%

Table A.1: Computational Results obtained with the node selection methods on the SpRand networks.

		$\mathcal{FLSM}$				$\mathcal{BDLSM}$			
		$\mathcal{FIFO}$	$\mathcal{SLLF}$	$\mathcal{LS}$	$\mathcal{THR}$	$\mathcal{FIFO}$	$\mathcal{SLLF}$	$\mathcal{LS}$	$\mathcal{THR}$
AVG=40	Time	87.14%	86.58%	85.39%	84.80%	45.07%	43.96%	25.79%	46.33%
	# Dom	8577.25	9083.29	12261.26	8608.54	1424.34	1421.93	1253.02	1395.43
	# Iter.	84.75%	84.01%	85.02%	84.24%	41.65%	41.73%	24.12%	44.92%
AVG=60	Time	94.25%	94.01%	93.55%	93.27%	64.44%	63.70%	51.73%	65.87%
	# Dom	25989.25	27781.44	39216.56	25744.51	4814.45	4831.27	4294.89	4966.03
	# Iter.	92.71%	92.97%	93.53%	93.00%	63.16%	63.16%	50.84%	63.76%
AVG=80	Time	93.13%	93.30%	93.24%	93.05%	63.87%	62.55%	53.70%	81.38%
	# Dom	59718.71	63961.79	97281.19	58913.69	11681.61	11740.36	12649.73	11670.65
	# Iter.	92.53%	92.90%	93.36%	93.11%	65.29%	63.38%	54.59%	77.95%
AVG=100	Time	94.45%	94.68%	94.54%	92.89%	61.85%	60.70%	47.40%	57.59%
	# Dom	136051.41	146532.26	234346.76	125485.26	12568.8	12596.21	12574.92	27263.42
	# Iter.	93.64%	93.27%	93.86%	93.19%	62.71%	61.72%	50.36%	55.79%

Table A.2: Computational Results obtained with the label selection methods on the SpRand networks.

		$\mathcal{FNSM}$							
		$\mathcal{SALS}$		$\mathcal{BLS}$		$\mathcal{SALF}$		$\mathcal{TOS}$	
		2^{nd}	3^{rd}	2^{nd}	3^{rd}	2^{nd}	3^{rd}	2^{nd}	3^{rd}
AVG=40	Time	55.97%	58.95%	88.65%	88.69%	46.19%	50.75%	26.96%	28.00%
	# Dom	12058.43	12460.56	12046.49	12447.88	9389.92	9774.86	7595.13	7935.29
	# Iter.	14.14%	16.91%	56.34%	59.98%	2.33%	7.48%	0.00%	0.00%
AVG=60	Time	77.29%	77.98%	94.67%	94.34%	66.51%	68.82%	47.20%	47.58%
	# Dom	36355.97	36757.51	36869.03	37271.04	28661.97	29012.42	23001.17	23341.08
	# Iter.	19.34%	21.72%	73.61%	74.68%	6.34%	10.82%	0.00%	0.00%
AVG=80	Time	82.03%	82.89%	95.82%	95.62%	71.16%	72.25%	45.01%	46.98%
	# Dom	83639.29	84150.17	86366.56	86873.34	66079.86	66529.5	51182.96	51608.35
	# Iter.	26.46%	27.27%	79.85%	80.17%	7.02%	10.81%	0.00%	0.00%
AVG=100	Time	88.03%	89.44%	95.77%	96.38%	79.15%	79.98%	54.81%	57.55%
	# Dom	167724.18	168559.43	175439.88	176280.39	127089.21	127767.31	106399.52	107170.95
	# Iter.	23.84%	33.04%	84.91%	84.71%	10.00%	12.73%	0.00%	0.00%

Table A.3: Computational Results obtained with the forward node selection methods on the SpAcyc networks.

		$\mathcal{BDNSM}$					
		$\mathcal{SALS}$		$\mathcal{BLS}$		$\mathcal{SALF}$	
		2^{nd}	3^{rd}	2^{nd}	3^{rd}	2^{nd}	3^{rd}
AVG=40	Time	1.13%	4.05%	14.68%	17.66%	16.77%	18.59%
	# Dom	2276	2549.59	2327.42	2577.51	2640.81	2905.97
	# Iter.	1.79%	18.78%	2.37%	16.21%	0.39%	2.92%
AVG=60	Time	2.42%	10.68%	34.17%	39.90%	25.99%	27.06%
	# Dom	8021.75	9749.05	8514.94	10624.74	8645.17	10059.9
	# Iter.	6.88%	30.47%	6.62%	30.75%	1.02%	1.96%
AVG=80	Time	25.02%	28.29%	75.68%	77.78%	54.02%	55.06%
	# Dom	22397.14	24392.58	23798.41	27198.08	22180.29	24441.48
	# Iter.	4.72%	36.68%	6.69%	13.84%	8.14%	9.37%
AVG=100	Time	29.00%	31.00%	77.61%	79.78%	58.27%	59.65%
	# Dom	47850.74	52182.2	51823.37	59364.96	51236.06	57169.43
	# Iter.	9.09%	42.64%	6.52%	15.46%	6.91%	8.75%

Table A.4: Computational Results obtained with the bi-directional node selection methods on the SpAcyc networks.

		$\mathcal{FLSM}$							
		$\mathcal{FIFO}$		$\mathcal{SLLF}$		$\mathcal{LS}$		$\mathcal{THR}$	
		2^{nd}	3^{rd}	2^{nd}	3^{rd}	2^{nd}	3^{rd}	2^{nd}	3^{rd}
AVG=40	Time	86.40%	88.41%	86.71%	88.52%	85.43%	87.84%	85.51%	86.86%
	# Dom	8575.56	8815.94	9081.75	9319.59	12259.51	12497.62	8606.69	8863.75
	# Iter.	84.13%	86.52%	84.40%	86.63%	84.86%	87.71%	84.48%	86.76%
AVG=60	Time	92.91%	93.12%	92.61%	92.53%	91.38%	92.15%	92.62%	93.43%
	# Dom	23362.76	23457.19	24906.84	24989.79	34351.69	34428.63	25740.44	25848.21
	# Iter.	90.90%	91.39%	91.26%	92.34%	91.58%	92.45%	93.27%	93.43%
AVG=80	Time	95.55%	96.33%	96.03%	95.62%	95.53%	95.51%	95.32%	95.24%
	# Dom	59583.36	59737.61	63818.54	63981.95	97101.85	97301.14	58798.72	58934.77
	# Iter.	95.06%	95.76%	95.25%	95.62%	95.65%	95.76%	95.79%	95.17%
AVG=100	Time	96.12%	96.16%	95.63%	95.68%	95.30%	96.00%	93.93%	94.59%
	# Dom	114943.62	121980.34	123128.4	131690.04	193935.2	213144.08	124982.16	125577.03
	# Iter.	95.24%	95.85%	95.01%	95.89%	96.07%	95.33%	95.86%	95.36%

Table A.5: Computational Results obtained with the forward label selection methods on the SpAcyc networks.

		$\mathcal{BDLSM}$							
		$\mathcal{FIFO}$		$\mathcal{SLLF}$		$\mathcal{LS}$		$\mathcal{THR}$	
		2^{nd}	3^{rd}	2^{nd}	3^{rd}	2^{nd}	3^{rd}	2^{nd}	3^{rd}
AVG=40	Time	46.24%	50.87%	45.63%	51.02%	27.33%	34.28%	46.17%	50.79%
	# Dom	1363.53	1550.43	1361.36	1562.64	1184.59	1326.16	1354.48	1540.99
	# Iter.	45.65%	50.46%	44.30%	49.64%	26.43%	33.78%	45.26%	50.05%
AVG=60	Time	64.90%	67.56%	64.29%	66.74%	53.28%	56.12%	65.94%	68.27%
	# Dom	4774.98	5025.03	4791.87	5010.57	4255.34	4150.03	4926.76	5011.9
	# Iter.	63.49%	66.24%	62.94%	65.46%	51.61%	54.29%	63.70%	67.55%
AVG=80	Time	61.84%	64.58%	61.28%	63.86%	51.15%	53.11%	65.06%	66.88%
	# Dom	11641.14	11864.59	11700.36	11984.39	12610.72	12962.48	11626.55	11936.23
	# Iter.	62.54%	64.22%	61.72%	64.10%	51.54%	53.73%	62.81%	65.04%
AVG=100	Time	73.02%	74.09%	72.44%	72.80%	62.79%	64.20%	76.64%	77.08%
	# Dom	26910.17	28268.01	27353.37	28536.7	27952.43	32265.27	27228.44	28431.11
	# Iter.	76.28%	76.56%	74.90%	76.53%	67.34%	68.78%	75.24%	76.19%

Table A.6: Computational Results obtained with the forward label selection methods on the SpAcyc networks.

	$\mathcal{FNSM}$						$\mathcal{BDNSM}$					
	$\mathcal{SALS}$		$\mathcal{BLS}$		$\mathcal{SALF}$		$\mathcal{SALS}$		$\mathcal{BLS}$		$\mathcal{SALF}$	
	CPU	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**
$R_1.I$	2.54	6.50	2.50	6.71	3.36	6.80	2.42	7.07	2.11	7.61	2.17	7.45
$R_2.I$	2.99	22.65	2.51	22.62	2.80	23.52	4.54	15.88	4.87	15.53	5.90	15.80
$R_3.I$	1.28	9.32	1.21	9.91	1.59	9.48	2.30	8.59	2.80	8.29	2.47	8.15
$R_4.I$	5.59	62.60	5.89	63.12	5.47	61.18	14.12	49.60	14.59	50.94	13.44	49.24
$R_5.I$	6.26	52.10	7.72	61.32	5.79	55.02	13.31	41.98	12.18	41.91	14.23	40.55
$R_6.I$	6.82	32.10	5.70	32.93	5.95	32.52	10.67	29.57	10.76	29.99	11.86	29.54
$R_7.I$	3.20	19.82	3.03	19.39	4.06	18.38	8.53	20.70	10.94	19.40	7.48	20.94
$R_8.I$	11.36	68.75	10.48	71.75	13.06	76.84	24.41	49.40	24.33	50.37	23.87	50.14
$R_9.I$	13.34	73.68	14.86	76.60	13.20	74.40	28.46	62.75	27.77	62.25	25.55	65.57
$R_{10}.I$	562.72	975.87	1023.95	2867.34	515.37	843.41	440.67	682.73	482.91	809.00	374.70	595.34
$R_{11}.I$	1.75	3.64	1.56	3.58	1.65	3.68	2.42	4.44	2.61	4.96	3.57	4.27
$R_{12}.I$	10.18	57.82	11.95	60.50	10.84	58.48	24.11	56.35	23.07	56.02	24.24	59.65
$R_{13}.I$	7.10	41.89	8.91	42.39	6.40	40.29	16.83	36.91	15.49	36.76	15.85	36.59
$R_{14}.I$	84.68	367.03	101.49	463.98	67.74	313.45	124.95	213.22	129.72	230.86	116.57	225.77
$R_{15}.I$	1564.99	1500.43	2940.62	4673.72	1448.74	1178.22	695.55	766.99	790.61	977.60	598.57	695.48
$R_1.II$	2.21	20.44	2.54	20.70	2.71	19.73	4.72	15.76	5.49	15.12	4.88	15.97
$R_2.II$	1.67	21.59	2.14	21.28	2.26	22.23	5.85	20.01	5.31	20.36	4.98	20.68
$R_3.II$	8.19	68.42	7.04	69.67	7.07	68.01	16.21	58.97	17.65	59.24	17.50	54.12
$R_4.II$	10.55	98.68	12.32	111.36	9.68	95.83	25.71	80.34	25.22	80.66	25.96	91.71
$R_5.II$	184.44	759.85	289.65	1593.47	159.04	669.03	275.92	571.03	244.72	569.15	227.15	458.07
$R_6.II$	9.09	56.03	10.90	63.40	8.45	53.22	17.67	37.49	16.65	37.66	16.76	36.08
$R_7.II$	5.59	33.69	6.27	34.44	5.51	33.82	14.87	30.73	13.26	30.69	14.14	30.09
$R_8.II$	14.74	79.75	12.73	81.34	12.56	75.16	23.72	52.65	24.80	53.21	23.00	52.53
$R_9.II$	84.79	394.01	110.51	556.35	76.16	399.56	138.43	255.75	139.54	272.49	131.50	265.45
$R_{10}.II$	2897.92	1442.01	6683.67	6769.88	2857.26	1232.00	1330.27	1351.01	1766.97	2450.52	1290.42	1215.73
$R_{11}.II$	19.82	113.20	23.67	123.42	22.34	111.69	38.16	73.52	38.28	73.63	36.85	74.84
$R_{12}.II$	48.40	234.18	55.87	261.71	47.95	237.28	97.12	191.29	96.27	193.57	88.61	182.07
$R_{13}.II$	30.82	149.25	33.52	164.49	31.99	151.38	65.23	126.17	64.63	130.16	61.32	123.16
$R_{14}.II$	68.58	300.29	81.89	374.66	63.78	290.31	110.99	178.93	111.90	177.81	101.02	173.85
$R_{15}.II$	95313.91	5297.87	222074.81	43150.34	91000.83	2103.36	2687.98	1872.40	4022.58	3142.01	2736.99	1145.62
$R_1.III$	9.66	87.48	9.49	88.32	10.92	98.78	17.76	52.57	18.86	51.43	17.46	54.47
$R_2.III$	2.69	25.75	2.28	24.39	2.23	25.02	5.90	15.53	5.36	15.17	5.64	15.56
$R_3.III$	6.87	61.27	6.27	64.58	6.09	60.96	16.51	57.94	17.51	57.20	15.69	58.36
$R_4.III$	4.62	45.97	4.09	46.57	4.31	44.06	13.17	45.44	13.91	45.72	13.17	46.53
$R_5.III$	299.83	885.00	543.40	2356.78	271.79	756.48	327.30	572.17	363.62	708.87	273.60	487.56
$R_6.III$	11.87	69.60	11.79	69.28	10.95	67.86	19.92	50.17	19.27	50.01	18.97	47.53
$R_7.III$	19.64	112.70	20.67	116.91	17.33	99.84	38.97	81.99	38.47	80.37	34.94	81.58
$R_8.III$	200.56	821.89	294.53	1345.86	170.55	748.98	314.42	501.24	306.91	520.15	272.79	469.94
$R_9.III$	510.70	1365.53	1011.79	3402.10	439.35	1150.89	708.37	1013.38	620.50	918.56	528.51	787.33
$R_{10}.III$	4241.05	2210.31	7701.28	8739.44	4014.56	1414.49	1427.09	1112.32	1982.81	1838.06	1348.38	904.13
$R_{11}.III$	56.25	261.15	65.42	307.42	51.06	245.21	115.30	202.11	114.24	209.49	105.06	199.67
$R_{12}.III$	15.97	74.34	15.34	75.15	14.02	75.54	31.31	66.17	32.52	66.63	33.24	68.25
$R_{13}.III$	59.50	273.10	84.17	395.17	57.78	269.55	109.07	212.29	109.89	213.05	103.68	198.36
$R_{14}.III$	1294.17	2039.33	2447.81	5926.37	1147.11	1672.09	1257.74	1198.58	1346.15	1462.83	1109.97	1046.59
$R_{15}.III$	28134845.75	13180.36	45271709.66	184644.54	27380407.28	3242.37	12328.20	2893.64	22312.75	8223.46	15713.71	1600.88
$R_1.IV$	0.94	49.49	7.75	53.97	0.24	55.80	11.14	37.85	11.62	37.46	12.15	37.88
$R_2.IV$	16.65	122.53	19.17	152.80	0.17	119.82	44.35	119.99	40.80	124.01	36.45	117.43
$R_3.IV$	16.60	38.29	6.89	38.28	0.85	39.31	7.10	30.37	10.18	30.75	10.87	31.37

continued on next page

continued from previous page

	$\mathcal{FNSM}$						$\mathcal{BDNSM}$					
	$\mathcal{SALS}$		$\mathcal{BLS}$		$\mathcal{SALF}$		$\mathcal{SALS}$		$\mathcal{BLS}$		$\mathcal{SALF}$	
	CPU	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**
$R_4.IV$	31.90	174.15	30.26	217.79	31.47	175.85	44.70	129.83	45.40	127.93	42.46	122.42
$R_5.IV$	26582.40	4456.12	38682.40	24714.32	24913.37	2340.49	2659.99	1528.32	4262.63	2703.46	2422.26	1204.58
$R_6.IV$	16.78	124.04	27.24	127.57	16.60	120.25	39.17	82.62	37.27	81.27	34.91	80.25
$R_7.IV$	250.96	871.51	343.31	1392.43	187.75	781.61	406.66	622.61	416.25	674.02	338.07	586.70
$R_8.IV$	78.96	320.09	78.68	360.64	63.53	302.14	110.65	210.36	110.44	211.44	102.81	202.70
$R_9.IV$	1467.62	1896.63	2806.62	5459.58	1311.91	1639.90	1093.78	950.59	1291.45	1291.32	929.92	921.38
$R_{10}.IV$	19013078.61	20899.72	32497687.87	384996.11	17835216.84	3960.57	12142.90	3854.02	20622.97	10258.33	25258.82	1883.90
$R_{11}.IV$	31.30	137.94	47.60	172.21	36.38	145.22	53.22	105.17	56.81	104.78	52.81	106.30
$R_{12}.IV$	141.94	576.61	218.70	796.42	142.90	524.13	218.80	357.71	218.82	359.69	210.66	368.45
$R_{13}.IV$	530.15	1379.78	842.06	2492.69	447.26	1181.03	697.39	802.59	729.32	943.00	644.36	777.96
$R_{14}.IV$	5913.74	3338.79	10717.24	13627.42	5460.25	2289.71	2783.92	2018.15	3500.23	3053.23	2271.08	1317.44
$R_{15}.IV$	OoM	OoM	OoM	OoM	OoM	OoM	59409.95	4519.60	284574.01	22035.16	156973.66	2285.45

Table A.7: Computational Results obtained with the node selection methods on the SpRand networks OoM indicates out of memory.

	$\mathcal{FNSM}$								$\mathcal{BDNSM}$					
	$\mathcal{SALS}$		$\mathcal{BLS}$		$\mathcal{SALF}$		$\mathcal{TOS}$		$\mathcal{SALS}$		$\mathcal{BLS}$		$\mathcal{SALF}$	
	CPU	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**
$AR_1.I$	1.75	11.66	2.67	11.32	3.51	11.74	227.01	2499.00	4.97	7.02	3.87	7.38	2.37	7.15
$AR_2.I$	3.98	22.94	2.31	22.30	1.69	23.79	183.35	1999.00	3.85	15.91	4.69	15.56	6.18	15.88
$AR_3.I$	1.87	9.74	0.28	9.90	2.38	9.98	136.29	1499.00	3.76	8.29	1.84	8.89	2.42	8.86
$AR_4.I$	4.46	62.54	7.25	63.73	15.21	61.46	89.96	999.00	12.54	49.17	12.85	49.93	11.01	52.61
$AR_5.I$	8.62	53.70	8.78	61.10	9.77	55.88	46.88	499.00	13.64	41.05	11.37	41.04	11.17	40.71
$AR_6.I$	3.13	32.42	5.68	32.05	8.97	32.44	376.74	2499.00	11.76	29.28	11.38	29.68	11.21	29.23
$AR_7.I$	2.66	19.19	1.09	19.76	6.54	18.93	301.59	1999.00	8.60	20.74	5.18	19.30	7.91	20.28
$AR_8.I$	13.51	67.33	10.82	69.99	24.49	74.03	225.19	1499.00	24.06	49.82	23.76	50.15	21.87	50.62
$AR_9.I$	15.61	73.56	15.41	76.92	22.97	74.58	152.90	999.00	23.65	60.12	28.37	60.83	22.36	64.23
$AR_{10}.I$	318.57	1220.55	434.20	1775.78	353.93	832.39	87.70	499.00	423.57	800.25	439.18	912.07	398.28	670.83
$AR_{11}.I$	0.56	3.79	0.34	3.61	1.76	3.42	450.72	2499.00	2.13	4.94	3.91	4.12	3.53	4.80
$AR_{12}.I$	13.39	55.88	11.79	57.20	16.65	56.06	361.35	1999.00	22.58	56.59	22.45	56.33	22.38	59.47
$AR_{13}.I$	8.75	40.40	7.94	41.14	15.21	40.13	274.08	1499.00	15.59	37.09	15.96	37.38	15.24	37.12
$AR_{14}.I$	86.13	364.86	95.39	404.62	114.54	309.64	186.12	999.00	117.96	219.62	117.49	223.85	110.43	220.88
$AR_{15}.I$	533.99	1741.22	668.67	2239.27	608.88	1169.83	107.47	499.00	581.67	969.66	619.25	1080.45	528.24	830.46
$AR_1.II$	2.70	18.50	1.09	18.88	1.86	17.47	226.88	2499.00	6.71	17.01	4.15	17.65	8.55	17.24
$AR_2.II$	1.97	21.04	3.41	21.89	3.91	22.27	178.57	1999.00	5.21	20.78	4.37	20.32	5.49	20.54
$AR_3.II$	10.33	64.77	9.98	66.80	13.51	69.77	138.27	1499.00	15.66	59.24	13.56	60.62	14.28	56.52
$AR_4.II$	13.83	97.86	13.13	109.42	20.08	99.15	91.62	999.00	21.04	79.78	24.16	79.27	23.82	88.30
$AR_5.II$	153.85	919.84	194.47	1273.43	179.80	704.42	51.37	499.00	208.32	464.74	206.00	483.91	190.53	425.29
$AR_6.II$	8.96	56.40	13.77	60.37	15.27	53.79	375.87	2499.00	14.63	37.49	16.07	37.05	15.65	36.27
$AR_7.II$	4.89	33.88	7.64	34.25	11.97	33.82	301.95	1999.00	13.93	30.07	13.98	30.52	12.65	30.90
$AR_8.II$	15.65	79.42	14.40	80.66	19.50	75.64	226.07	1499.00	24.04	52.99	23.91	52.88	22.15	52.41
$AR_9.II$	86.58	406.85	108.08	519.45	131.40	402.27	155.83	999.00	129.84	248.48	126.28	251.38	121.82	254.96
$AR_{10}.II$	636.36	2169.16	916.65	3391.43	658.55	1297.02	101.90	499.00	946.12	1362.50	1034.62	1502.46	832.82	1037.13
$AR_{11}.II$	24.49	116.19	26.70	123.37	40.89	118.36	450.80	2499.00	37.49	75.11	36.43	73.38	36.30	74.95
$AR_{12}.II$	53.24	238.58	58.43	259.43	88.63	245.44	364.85	1999.00	90.62	187.75	88.73	190.00	84.20	180.54
$AR_{13}.II$	32.34	152.88	37.84	161.75	58.48	152.59	273.15	1499.00	59.41	126.39	61.76	128.37	59.88	124.04
$AR_{14}.II$	74.71	313.59	85.33	360.58	108.88	288.36	185.99	999.00	109.18	192.51	108.94	197.13	103.31	184.83
$AR_{15}.II$	2617.43	7896.97	3453.25	11167.66	1755.55	2245.93	144.78	499.00	2033.49	3283.27	2317.82	4288.49	1372.12	1579.12
$AR_1.III$	3.56	32.14	5.94	32.37	5.39	31.13	225.63	2499.00	8.93	28.67	9.27	28.99	6.56	27.35
$AR_2.III$	2.86	25.60	5.58	24.33	8.49	25.92	182.49	1999.00	4.80	15.58	6.33	15.96	5.52	15.87
$AR_3.III$	6.73	61.82	6.56	64.50	12.05	60.13	137.89	1499.00	14.33	54.28	13.53	54.91	14.02	55.38
$AR_4.III$	4.10	45.46	6.99	46.84	9.99	44.71	93.55	999.00	11.79	45.84	12.55	45.88	10.79	46.21
$AR_5.III$	194.42	1051.59	265.15	1572.48	222.66	750.77	53.04	499.00	291.89	649.16	304.68	783.40	237.04	540.51
$AR_6.III$	12.84	68.92	13.19	68.69	16.77	67.94	376.34	2499.00	18.19	50.07	18.03	50.37	16.00	47.65
$AR_7.III$	21.98	112.86	22.18	116.48	29.68	101.75	303.18	1999.00	36.27	79.74	35.89	79.22	33.35	81.44
$AR_8.III$	209.08	907.85	253.82	1153.05	264.15	760.42	230.35	1499.00	273.67	467.38	266.74	456.03	252.95	444.45
$AR_9.III$	421.34	1637.98	567.32	2338.67	471.09	1169.86	161.45	999.00	564.00	892.41	490.65	749.08	447.63	708.35
$AR_{10}.III$	845.78	2769.52	1223.77	4398.79	801.89	1394.55	103.99	499.00	891.57	1170.34	1045.40	1501.93	776.27	964.86
$AR_{11}.III$	62.18	261.80	68.52	289.29	85.19	246.34	451.47	2499.00	116.67	214.80	112.23	213.50	102.49	204.08
$AR_{12}.III$	17.89	73.63	15.51	73.62	28.14	75.28	362.99	1999.00	29.82	66.03	30.52	66.61	28.37	67.92
$AR_{13}.III$	64.15	276.60	79.12	351.74	97.70	268.58	275.38	1499.00	101.04	200.35	104.91	201.99	97.70	199.21
$AR_{14}.III$	855.16	2579.08	1190.46	3942.17	903.40	1765.21	200.86	999.00	978.66	1153.12	983.74	1310.44	877.89	997.04
$AR_{15}.III$	6836.08	18041.88	9736.39	26768.17	4168.88	2682.51	223.87	499.00	5451.25	8039.26	6453.43	11072.07	3082.08	1982.18
$AR_1.IV$	14.08	129.33	18.89	143.55	23.12	129.47	225.58	2499.00	29.45	96.43	28.94	96.56	30.99	101.28
$AR_2.IV$	15.33	123.07	15.97	141.66	21.76	122.03	179.11	1999.00	32.43	117.87	34.35	122.18	30.39	114.66
$AR_3.IV$	4.62	38.47	4.95	38.73	3.47	39.63	134.76	1499.00	9.37	30.41	5.15	30.29	9.14	31.32

continued on next page

continued from previous page

	$\mathcal{FNSM}$								$\mathcal{BDNSM}$					
	$\mathcal{SALS}$		$\mathcal{BLS}$		$\mathcal{SALF}$		$\mathcal{TOS}$		$\mathcal{SALS}$		$\mathcal{BLS}$		$\mathcal{SALF}$	
	CPU	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**
$AR_4.IV$	24.18	183.77	25.59	217.96	35.37	176.17	93.01	999.00	40.94	128.63	40.19	124.68	38.39	121.10
$AR_5.IV$	1613.82	7842.65	2069.00	11010.41	1274.49	2452.52	89.79	499.00	1508.60	2712.07	1568.16	3532.78	1155.36	1588.99
$AR_6.IV$	25.59	126.97	27.94	127.74	34.34	120.96	375.72	2499.00	36.00	81.12	34.49	81.51	33.08	82.12
$AR_7.IV$	204.64	898.75	253.07	1176.24	259.51	793.17	306.13	1999.00	371.00	631.07	368.31	640.70	304.68	577.16
$AR_8.IV$	65.28	322.19	70.27	351.64	94.71	305.08	228.45	1499.00	106.76	206.72	104.19	211.04	96.28	202.77
$AR_9.IV$	728.77	2529.25	1013.92	3716.17	801.71	1769.29	170.05	999.00	919.45	1077.65	964.91	1229.64	763.12	1087.27
$AR_{10}.IV$	8562.58	24680.46	12181.97	38837.02	5097.71	3120.96	241.59	499.00	6759.80	12075.35	8442.29	17011.97	4028.56	2664.40
$AR_{11}.IV$	31.56	141.87	38.13	166.07	50.09	145.44	451.17	2499.00	49.88	103.35	49.68	103.07	48.20	105.10
$AR_{12}.IV$	145.78	600.10	169.60	712.12	198.93	531.40	364.26	1999.00	200.83	340.72	197.77	338.44	195.61	356.39
$AR_{13}.IV$	440.65	1529.76	575.73	2090.06	522.50	1171.10	281.26	1499.00	601.43	740.81	610.99	796.73	570.63	742.36
$AR_{14}.IV$	1719.51	4518.16	2487.80	7285.00	1653.21	2444.48	217.36	999.00	1888.02	2081.56	2141.00	2866.52	1590.53	1555.14
$AR_{15}.IV$	18150.07	42892.13	25975.18	64696.32	9908.05	3155.04	388.75	499.00	13853.29	18573.54	19881.86	31502.95	7078.72	2592.98

Table A.8: Computational Results obtained with the node selection methods on the SpAcyc networks.

	$\mathcal{FLSM}$								$\mathcal{BDLSM}$							
	$\mathcal{FIFO}$		$\mathcal{SLLF}$		$\mathcal{LS}$		$\mathcal{THR}$		$\mathcal{FIFO}$		$\mathcal{SLLF}$		$\mathcal{LS}$		$\mathcal{THR}$	
	CPU	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**
$R_1.I$	1.66	6.72	2.69	6.45	2.56	6.71	0.44	6.61	80.58	94.06	79.89	94.40	79.66	94.97	80.19	94.08
$R_2.I$	3.30	26.43	3.19	26.13	4.53	26.40	0.84	26.12	147.09	172.95	153.01	172.57	149.01	172.34	146.65	172.37
$R_3.I$	1.90	9.76	1.77	9.48	1.51	9.30	0.22	9.33	78.29	90.43	81.01	90.18	78.10	90.96	77.69	90.98
$R_4.I$	10.60	69.29	10.53	69.89	10.80	69.38	16.97	70.02	302.13	340.28	343.25	382.70	352.03	398.93	266.45	311.38
$R_5.I$	11.03	71.00	11.19	71.05	11.99	71.33	15.56	74.01	187.88	201.98	189.68	193.07	181.29	193.84	128.68	143.65
$R_6.I$	7.32	34.48	8.56	34.05	9.17	34.13	15.51	34.66	395.39	279.01	395.17	279.30	395.05	279.99	393.14	283.61
$R_7.I$	4.87	19.37	4.73	19.43	5.17	19.02	0.75	19.54	285.79	200.47	287.20	200.23	284.74	200.82	282.89	200.55
$R_8.I$	19.92	85.11	19.10	85.04	19.01	85.58	16.28	89.07	480.55	328.19	505.26	346.48	489.64	335.45	420.96	298.49
$R_9.I$	21.14	86.46	21.54	86.35	21.20	86.97	16.45	88.81	560.63	380.68	529.06	359.74	528.33	359.75	517.12	365.48
$R_{10}.I$	588.07	2119.28	592.87	2126.71	601.14	2119.05	1685.36	6261.87	5694.55	3608.18	5559.57	3513.90	4974.66	3108.70	8282.79	5725.02
$R_{11}.I$	0.79	3.46	0.68	3.64	1.76	3.24	0.01	3.05	145.68	86.97	144.00	86.94	144.09	86.40	144.70	86.65
$R_{12}.I$	19.86	69.73	19.95	69.89	20.05	69.69	15.39	71.66	702.88	411.90	704.57	411.77	705.48	411.34	708.64	423.86
$R_{13}.I$	12.18	45.29	12.24	45.54	14.66	45.98	15.21	45.94	316.46	179.66	310.14	175.51	314.15	179.85	212.27	125.79
$R_{14}.I$	129.38	442.63	131.08	444.50	126.42	442.33	172.35	661.40	2019.45	1133.91	2021.32	1127.65	1701.23	946.84	2018.75	1185.40
$R_{15}.I$	950.91	2865.59	960.63	2884.31	971.98	2865.15	4352.23	12462.33	8286.88	4363.77	8663.89	4548.24	7396.80	3860.25	13150.64	7597.63
$R_1.II$	2.50	20.25	2.71	20.17	3.97	20.78	16.56	20.35	240.76	281.11	239.12	281.35	241.29	281.73	236.20	281.13
$R_2.II$	3.04	22.69	3.52	22.77	2.82	22.90	0.45	22.08	256.92	298.48	258.38	298.67	256.34	298.99	254.13	298.76
$R_3.II$	10.03	74.25	11.71	75.27	11.64	74.27	15.64	76.79	523.33	607.49	523.38	607.06	524.83	607.60	424.14	503.23
$R_4.II$	18.33	126.14	18.27	126.84	18.66	126.21	16.48	129.69	446.55	506.88	456.75	514.97	447.01	502.93	458.41	538.85
$R_5.II$	264.70	1580.93	265.67	1586.25	270.37	1580.95	452.13	2939.30	2955.77	3184.31	2891.31	3115.23	2673.05	2844.92	3295.34	3784.41
$R_6.II$	16.22	72.23	16.52	72.59	16.09	72.73	15.89	74.86	346.21	238.34	347.16	238.01	346.76	238.85	252.32	182.46
$R_7.II$	9.20	39.73	9.83	39.13	9.71	39.57	0.14	39.34	494.46	345.51	494.24	345.86	491.08	345.82	189.36	134.92
$R_8.II$	21.99	87.71	20.81	87.40	22.50	87.56	16.37	92.27	463.33	317.99	480.44	330.43	515.51	355.63	339.17	239.35
$R_9.II$	169.58	670.99	169.42	670.04	170.04	670.25	203.07	852.97	2295.63	1545.47	2396.15	1619.24	2039.49	1361.61	2248.86	1582.63
$R_{10}.II$	1217.63	4088.44	1215.88	4103.29	1259.78	4088.65	6770.63	19650.06	13853.72	8899.83	13719.05	8815.73	13581.79	8591.84	21014.32	14340.53
$R_{11}.II$	36.33	134.43	37.79	134.59	36.71	134.19	31.59	141.41	915.90	532.02	923.17	537.86	948.12	550.39	850.48	509.92
$R_{12}.II$	87.28	303.89	87.37	305.03	85.66	303.28	94.78	340.57	2724.71	1585.14	2624.64	1530.82	2414.36	1404.63	2151.20	1286.83
$R_{13}.II$	59.19	205.57	60.80	208.81	57.24	205.62	47.12	220.23	1472.96	847.87	1391.76	799.17	1485.54	851.91	1440.22	853.82
$R_{14}.II$	137.32	450.97	137.84	450.32	136.48	450.69	156.25	552.26	2109.12	1185.19	2110.60	1185.03	1811.34	1003.10	1826.99	1074.31
$R_{15}.II$	3160.42	9164.04	3180.85	9213.02	3389.32	9164.85	165173.75	121638.28	24339.81	12783.16	24226.99	12741.00	21308.95	11035.25	69723.85	40079.70
$R_1.III$	18.62	130.36	19.10	130.51	18.34	130.68	16.50	130.41	641.96	745.40	640.48	745.16	640.00	745.18	627.91	745.54
$R_2.III$	3.39	26.11	3.65	26.65	3.12	26.98	0.37	26.06	132.03	153.99	185.09	215.97	185.46	215.09	131.63	153.72
$R_3.III$	10.16	70.80	10.25	70.26	10.65	70.97	0.96	71.59	581.26	672.32	586.41	672.50	633.13	733.28	500.09	592.96
$R_4.III$	7.74	48.39	7.10	48.96	6.57	48.41	16.56	48.03	313.39	354.53	313.80	354.18	314.70	354.86	241.75	277.11
$R_5.III$	375.64	2243.26	375.36	2249.61	389.74	2243.42	827.07	5296.35	3985.29	4276.52	3857.15	4130.29	3377.71	3557.64	4386.32	5061.84
$R_6.III$	16.42	69.82	15.82	69.87	16.49	69.66	32.33	72.67	575.76	403.57	576.36	403.22	595.22	416.73	225.57	160.35
$R_7.III$	33.15	142.34	33.45	142.72	34.42	142.04	31.61	152.63	1131.32	791.98	1076.12	753.97	974.10	680.33	774.86	558.76
$R_8.III$	381.99	1576.94	382.86	1579.80	384.31	1576.65	515.10	2302.36	5581.99	3807.52	5699.24	3895.76	4532.08	3047.27	4516.48	3220.18
$R_9.III$	705.15	2776.34	707.96	2788.06	720.45	2776.24	1482.59	6023.20	8206.65	5518.01	8842.31	5966.96	7300.70	4857.81	8547.54	6061.38
$R_{10}.III$	1548.85	5538.27	1555.80	5565.12	1645.33	5538.59	8752.48	23810.91	14970.49	9590.91	15291.44	9807.61	13168.97	8294.98	23425.93	16185.07
$R_{11}.III$	112.69	400.32	112.43	400.90	111.40	400.50	124.93	425.39	3512.00	2064.38	3828.94	2254.96	3153.38	1843.80	3196.30	1917.74
$R_{12}.III$	24.51	85.18	24.35	85.90	23.70	85.30	15.53	89.65	878.21	516.69	883.65	516.80	886.80	516.80	551.17	327.01
$R_{13}.III$	124.99	430.71	125.74	431.67	122.27	430.85	156.35	570.33	2295.19	1320.49	2527.66	1460.76	2309.09	1323.43	2153.74	1281.88
$R_{14}.III$	1358.22	4418.71	1365.98	4433.72	1407.85	4418.77	3463.24	11012.83	16838.03	9430.14	16502.24	9215.14	15101.85	8318.22	19568.37	11573.86
$R_{15}.III$	7997.50	21426.95	8218.84	21591.44	9647.89	21426.59	12358679.68	912772.33	57779.48	29963.18	57797.37	30058.77	53029.66	26677.65	314894.58	170976.28
$R_1.IV$	8.56	61.87	8.51	61.34	8.09	61.24	16.83	61.36	483.68	564.11	494.55	580.92	476.52	556.49	474.11	565.00
$R_2.IV$	27.08	195.74	27.40	195.24	27.24	195.31	47.59	216.84	1620.58	1889.04	1621.16	1889.61	1633.79	1889.68	1902.41	2255.10
$R_3.IV$	7.04	53.13	7.22	53.01	8.71	53.67	0.77	54.19	367.11	423.36	366.81	423.25	366.10	423.18	219.43	259.94

continued on next page

continued from previous page

	$\mathcal{FLSM}$								$\mathcal{BDLSM}$							
	$\mathcal{FIFO}$		$\mathcal{SLLF}$		$\mathcal{LS}$		$\mathcal{THR}$		$\mathcal{FIFO}$		$\mathcal{SLLF}$		$\mathcal{LS}$		$\mathcal{THR}$	
	CPU	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**
$R_4.IV$	40.07	270.75	40.13	270.14	41.41	270.40	47.90	290.99	857.89	965.08	871.86	986.66	822.95	920.19	733.77	858.38
$R_5.IV$	1817.30	9940.90	1838.00	9984.52	2046.02	9940.96	35272.60	61206.46	15521.88	16514.00	15567.89	16533.68	13523.88	13857.31	24519.29	28025.67
$R_6.IV$	38.48	162.26	38.79	163.70	37.96	162.12	47.11	168.99	1164.75	819.17	1171.88	819.35	1163.57	817.13	1090.62	783.33
$R_7.IV$	329.04	1401.88	331.16	1408.51	332.53	1401.97	453.91	1995.59	7525.21	5240.00	7088.61	4928.58	7409.84	5116.40	8484.38	6087.46
$R_8.IV$	106.74	440.37	106.66	440.63	106.51	440.38	156.83	482.24	2387.98	1646.68	2432.58	1679.58	2074.83	1415.41	1747.31	1242.16
$R_9.IV$	1273.94	4955.72	1281.12	4981.86	1335.41	4955.36	3510.94	12489.77	13502.48	8973.70	13219.94	8774.02	12341.06	8070.60	14527.47	10273.61
$R_{10}.IV$	8607.82	26365.68	8906.02	26603.92	10857.65	26365.65	36388732.73	1574784.19	61302.16	37532.40	62215.77	38196.46	54733.69	32150.37	972851.92	448350.66
$R_{11}.IV$	49.23	179.66	49.89	179.70	49.79	179.62	47.79	193.79	1788.42	1050.29	1790.78	1050.31	1790.05	1050.27	1745.75	1045.28
$R_{12}.IV$	236.14	827.58	236.97	829.81	235.99	827.19	281.99	1088.83	5059.51	2929.50	5048.41	2925.12	3674.15	2096.14	3946.90	2361.16
$R_{13}.IV$	784.86	2679.73	786.10	2687.19	798.01	2679.70	1170.32	4183.32	12855.93	7360.32	12841.46	7358.96	9744.19	5488.01	13262.54	7904.79
$R_{14}.IV$	2737.04	8904.86	2757.27	8945.47	2953.32	8904.77	13322.41	32828.50	31597.72	17642.74	30578.20	17052.86	26134.11	14213.89	40716.31	24078.43
$R_{15}.IV$	17807.97	41502.67	18791.16	42018.93	24075.49	41502.94	OoM	OoM	111076.74	55845.38	109012.31	54990.51	106302.28	51010.87	OoM	OoM

Table A.9: Computational Results obtained with the label selection methods on the SpRand networks OoM indicates out of memory.

	$\mathcal{FLSM}$								$\mathcal{BDLSM}$							
	$\mathcal{FIFO}$		$\mathcal{SLLF}$		$\mathcal{LS}$		$\mathcal{THR}$		$\mathcal{FIFO}$		$\mathcal{SLLF}$		$\mathcal{LS}$		$\mathcal{THR}$	
	CPU	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**
$AR_1.I$	0.20	11.96	2.41	11.21	2.92	11.45	2.91	11.60	85.31	99.71	91.85	107.54	93.08	107.46	89.08	107.88
$AR_2.I$	3.33	26.35	4.09	26.26	2.57	25.04	1.31	26.07	148.99	172.23	146.78	172.67	151.98	172.76	144.31	172.36
$AR_3.I$	1.54	9.36	1.93	9.09	0.98	9.92	2.87	9.70	78.27	90.51	78.46	90.92	78.58	90.07	76.17	90.88
$AR_4.I$	10.41	69.04	10.36	69.45	12.97	69.47	12.35	69.77	298.67	338.36	324.36	372.81	344.57	384.48	259.32	301.38
$AR_5.I$	11.63	70.24	9.03	69.68	11.09	69.41	9.56	70.99	187.52	201.54	178.27	193.62	191.39	193.04	176.29	199.27
$AR_6.I$	6.78	33.35	6.97	33.01	7.39	33.76	8.37	33.21	394.74	279.23	393.93	279.14	400.26	279.31	388.86	279.63
$AR_7.I$	6.57	19.87	3.14	19.90	4.82	19.64	6.89	19.21	284.95	200.31	284.95	200.48	288.89	200.72	280.08	200.72
$AR_8.I$	19.77	80.15	18.38	81.05	17.23	79.07	17.47	82.90	527.11	361.47	532.29	365.63	541.27	359.97	461.08	328.50
$AR_9.I$	19.27	83.29	21.62	83.89	22.67	83.73	20.45	83.67	548.33	377.84	525.76	360.78	542.22	359.04	518.39	366.52
$AR_{10}.I$	427.87	1536.13	412.23	1474.76	400.99	1409.40	373.21	1575.20	4690.46	2953.15	4551.58	2864.50	4441.17	2566.33	4268.76	2912.10
$AR_{11}.I$	1.39	3.55	1.31	3.82	2.21	3.21	0.43	3.93	143.74	86.15	145.81	86.39	145.20	86.02	144.11	86.49
$AR_{12}.I$	18.12	65.11	19.43	65.01	17.75	65.85	17.21	65.22	702.25	411.11	700.64	410.57	713.23	410.56	690.19	411.49
$AR_{13}.I$	11.51	42.25	9.60	42.63	12.18	42.06	10.52	42.03	322.94	184.77	321.41	184.57	334.98	184.07	250.38	148.66
$AR_{14}.I$	116.01	407.49	118.17	405.84	147.13	399.68	109.88	405.47	1872.90	1048.27	1857.65	1040.89	1678.48	878.93	1795.20	1055.79
$AR_{15}.I$	647.14	1946.65	649.44	1940.28	603.27	1779.51	545.67	1912.07	6562.64	3446.94	6442.20	3382.69	6129.55	2943.76	6050.22	3450.73
$AR_1.II$	3.40	20.21	3.57	20.61	5.77	20.03	1.21	20.79	304.22	360.17	304.70	360.76	309.09	360.29	290.13	347.17
$AR_2.II$	2.97	22.89	5.56	22.49	4.84	22.10	2.97	22.33	254.37	298.86	258.62	298.98	256.94	298.35	253.64	298.40
$AR_3.II$	10.02	70.14	11.58	68.23	10.48	64.84	10.83	66.34	558.84	649.17	550.46	644.22	556.27	640.53	520.32	619.19
$AR_4.II$	18.79	119.08	17.64	121.18	18.00	117.77	17.83	117.22	468.39	533.33	471.58	535.31	472.90	521.98	465.84	546.51
$AR_5.II$	205.49	1236.13	202.43	1222.96	203.22	1191.90	179.83	1253.98	2663.55	2845.79	2578.67	2762.26	2481.48	2488.05	2445.35	2783.33
$AR_6.II$	13.46	58.75	13.70	57.39	12.69	57.85	12.39	57.47	523.65	364.82	520.27	363.36	532.22	362.86	510.92	365.01
$AR_7.II$	7.98	39.33	9.81	39.48	6.81	39.81	7.36	39.51	492.84	345.20	491.69	345.31	499.27	345.12	482.95	344.59
$AR_8.II$	17.37	82.88	18.28	80.08	17.99	80.69	19.57	81.11	461.55	317.56	479.54	330.69	519.87	346.80	347.05	246.02
$AR_9.II$	153.81	612.07	148.76	602.78	151.00	593.03	137.68	611.46	2187.81	1472.83	2278.02	1538.06	2063.25	1313.53	1980.08	1393.93
$AR_{10}.II$	823.58	2781.48	818.85	2762.53	771.79	2556.36	676.36	2761.98	10352.75	6555.85	10182.00	6452.11	10339.32	6156.81	9719.16	6595.28
$AR_{11}.II$	35.72	126.82	34.31	125.78	34.56	125.80	33.07	126.76	930.19	541.31	930.91	541.91	966.52	545.39	903.37	541.93
$AR_{12}.II$	75.13	275.06	70.15	262.43	73.35	260.73	69.81	270.61	2769.84	1619.72	2727.26	1595.80	2668.69	1525.60	2613.66	1561.99
$AR_{13}.II$	55.48	190.47	56.07	196.71	54.40	189.04	49.01	192.12	1423.57	820.28	1353.34	778.18	1476.10	823.62	1331.94	791.97
$AR_{14}.II$	123.16	411.53	123.27	413.14	123.05	407.26	111.36	410.67	2041.46	1148.18	2039.06	1147.70	1848.00	971.22	1728.48	1016.58
$AR_{15}.II$	1802.42	5174.11	1798.11	5156.48	1598.07	4483.17	1523.43	5119.58	16307.94	8480.25	16046.86	8349.74	15254.51	7242.78	15492.24	8777.65
$AR_1.III$	6.24	33.01	3.09	33.28	5.33	33.58	3.63	33.95	418.40	495.04	420.35	495.91	423.23	495.26	415.54	495.24
$AR_2.III$	4.28	26.52	3.80	26.50	4.68	26.38	3.32	26.68	131.58	153.25	185.83	215.68	189.58	215.76	127.99	153.78
$AR_3.III$	9.91	69.51	10.48	69.10	10.09	69.07	10.32	69.40	541.26	627.16	543.75	629.85	591.60	679.86	581.47	689.94
$AR_4.III$	4.91	48.50	5.78	48.72	6.88	48.85	6.42	48.95	311.66	353.25	311.82	353.70	316.67	353.75	302.83	352.08
$AR_5.III$	290.00	1743.75	288.52	1726.29	288.13	1668.53	249.21	1755.95	3492.37	3729.31	3342.81	3557.26	3223.92	3169.54	3219.34	3668.34
$AR_6.III$	15.00	69.32	15.70	69.08	15.21	68.64	15.94	69.17	575.47	403.83	574.85	403.93	610.76	416.03	373.62	267.36
$AR_7.III$	31.87	141.25	31.69	138.61	31.59	138.30	31.60	141.91	1128.13	791.53	1076.37	753.44	1000.95	680.12	936.49	670.96
$AR_8.III$	338.21	1420.28	331.22	1391.33	328.04	1357.28	311.57	1406.85	5237.44	3563.31	5011.48	3408.97	4648.71	2956.71	4628.11	3291.05
$AR_9.III$	589.29	2343.63	582.57	2319.33	570.50	2224.61	529.73	2342.19	7447.68	4993.65	7934.39	5324.09	6645.33	4126.74	6502.09	4588.81
$AR_{10}.III$	1014.57	3645.35	1009.10	3600.36	927.82	3195.89	855.90	3555.85	10233.01	6476.52	10165.44	6433.80	9426.97	5506.59	9387.79	6409.03
$AR_{11}.III$	92.56	333.77	92.76	333.74	92.01	328.81	87.60	334.92	4062.15	2392.40	4137.88	2438.89	4099.27	2368.49	3740.40	2243.25
$AR_{12}.III$	25.85	85.39	24.66	85.31	26.54	85.96	21.86	85.63	868.19	508.43	866.21	508.74	888.28	508.44	841.94	502.93
$AR_{13}.III$	114.82	395.36	113.76	393.97	114.83	391.16	103.24	396.99	2193.97	1260.77	2377.57	1372.68	2237.80	1238.53	1987.93	1182.75
$AR_{14}.III$	1094.19	3570.52	1071.37	3488.71	1047.62	3316.98	968.95	3537.41	13779.11	7647.39	13256.27	7352.20	13595.34	7018.43	12374.64	7273.64
$AR_{15}.III$	3688.85	9720.08	3668.31	9590.77	3078.34	7860.20	3092.62	9547.09	32097.60	16423.60	31495.45	16160.21	29109.02	13663.01	28992.04	16172.28
$AR_1.IV$	22.50	155.49	21.69	154.90	19.59	153.73	20.00	155.68	1097.81	1287.36	1243.27	1462.85	1308.35	1512.62	1050.17	1255.44
$AR_2.IV$	24.40	176.77	25.82	169.85	23.64	169.83	23.66	176.42	1472.08	1720.68	1451.40	1696.69	1466.97	1692.21	1445.26	1719.11
$AR_3.IV$	7.11	53.13	5.38	53.63	8.28	53.19	7.73	53.38	360.59	420.66	362.24	420.93	365.50	420.33	263.66	311.42

continued on next page

continued from previous page

	$\mathcal{FLSM}$								$\mathcal{BDLSM}$							
	$\mathcal{FIFO}$		$\mathcal{SLLF}$		$\mathcal{LS}$		$\mathcal{THR}$		$\mathcal{FIFO}$		$\mathcal{SLLF}$		$\mathcal{LS}$		$\mathcal{THR}$	
	CPU	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**	**CPU**	**# Iter.**
$AR_4.IV$	36.68	261.40	36.32	258.12	38.02	257.06	34.60	261.57	828.92	936.10	859.20	970.26	841.65	913.93	778.20	911.18
$AR_5.IV$	1207.91	6564.78	1191.62	6445.90	1160.66	5914.02	1010.76	6532.18	11376.86	11824.28	11383.51	11832.68	10578.51	9965.59	9840.27	11056.95
$AR_6.IV$	36.89	157.55	37.64	158.07	35.29	156.20	35.76	159.17	1418.60	1004.16	1420.14	1004.63	1447.40	1004.02	1389.12	1001.56
$AR_7.IV$	297.03	1280.23	295.10	1275.75	293.04	1240.13	280.17	1284.47	7844.85	5457.92	7797.91	5425.37	8153.88	5499.65	7250.21	5187.12
$AR_8.IV$	97.11	409.07	96.16	405.50	96.91	401.85	91.73	407.31	2350.40	1620.98	2369.69	1633.08	2057.82	1347.84	1919.58	1366.71
$AR_9.IV$	1008.44	3947.47	989.29	3859.10	957.84	3605.01	898.38	3914.38	12163.88	8063.15	11937.26	7911.09	11573.60	7069.93	10427.63	7347.47
$AR_{10}.IV$	3887.01	11708.90	3823.15	11522.22	3240.88	9387.75	3315.71	11641.05	34474.81	20889.41	33735.96	20525.44	29965.77	16652.34	29681.65	19642.82
$AR_{11}.IV$	75.55	176.43	76.03	176.07	47.38	173.07	46.11	177.00	1784.52	1047.29	1784.22	1047.42	1818.42	1047.77	1735.46	1040.30
$AR_{12}.IV$	219.84	780.07	224.60	785.41	216.61	764.90	207.37	781.40	4674.92	2703.47	4645.76	2685.85	3645.05	1978.63	4142.64	2474.73
$AR_{13}.IV$	677.65	2347.41	673.87	2326.70	659.77	2240.79	628.82	2343.08	12042.73	6873.51	11860.66	6770.43	9902.52	5288.08	10443.47	6205.09
$AR_{14}.IV$	1999.66	6482.39	1954.94	6341.80	1901.13	5899.03	1792.64	6445.77	24902.30	13769.00	23884.36	13205.50	23138.77	11789.90	21948.59	12875.82
$AR_{15}.IV$	6743.37	15636.20	6728.91	15795.50	5231.67	12058.31	5426.29	14746.04	52598.95	26053.09	51682.02	25726.70	47254.85	21598.11	47207.39	25646.87

Table A.10: Computational Results obtained with the label selection methods on the SpRand networks OoM indicates out of memory.

	$\mathcal{RLA}$			$\mathcal{RLA}$	
	CPU	**# Iter.**		**CPU**	**# Iter.**
$R_1.I$	1942.38	7316	$AR_1.I$	1250.03	5491
$R_2.I$	920.69	6289	$AR_2.I$	1422.33	6289
$R_3.I$	905	6234	$AR_3.I$	1265.89	6234
$R_4.I$	1685.97	8964	$AR_4.I$	1875.67	8964
$R_5.I$	2871.92	19420	$AR_5.I$	4110.38	19420
$R_6.I$	2447.79	10129	$AR_6.I$	3453.34	10129
$R_7.I$	2917.62	11826	$AR_7.I$	3984.89	11826
$R_8.I$	3946.21	13889	$AR_8.I$	4735.49	13889
$R_9.I$	7300.85	20638	$AR_9.I$	7141.39	20638
$R_{10}.I$	12569.05	298053	$AR_{10}.I$	114750.67	298053
$R_{11}.I$	3095.92	10899	$AR_{11}.I$	4391.38	10899
$R_{12}.I$	4264.64	14775	$AR_{12}.I$	5938.46	14775
$R_{13}.I$	4509.75	15629	$AR_{13}.I$	6313.01	15629
$R_{14}.I$	13448.37	46128	$AR_{14}.I$	18906.64	46128
$R_{15}.I$	16345.49	463594	$AR_{15}.I$	222937.61	463594
$R_1.II$	1451.35	9531	$AR_1.II$	1860.03	9249
$R_2.II$	1373.22	9504	$AR_2.II$	1372.1	9504
$R_3.II$	2427.51	14999	$AR_3.II$	3292.98	14999
$R_4.II$	3447.93	23434	$AR_4.II$	3151.91	23434
$R_5.II$	16754.67	161154	$AR_5.II$	24838.74	161154
$R_6.II$	4898.86	16164	$AR_6.II$	4495.99	16164
$R_7.II$	4556.05	18896	$AR_7.II$	4118.9	18896
$R_8.II$	8627.75	35658	$AR_8.II$	7815.4	35658
$R_9.II$	26255.2	104927	$AR_9.II$	23978.46	104927
$R_{10}.II$	28849.2	1515351	$AR_{10}.II$	26973.79	26004
$R_{11}.II$	10828	26004	$AR_{11}.II$	7238.23	27665
$R_{12}.II$	11313	27665	$AR_{12}.II$	16708.79	62787
$R_{13}.II$	25938.48	62787	$AR_{13}.II$	70637.12	243752
$R_{14}.II$	34265.07	243752	$AR_{14}.II$	70637.93	243752
$R_{15}.II$	39909.61	2449748	$AR_{15}.II$	132941.97	2449748
$R_1.III$	2247.16	14941	$AR_1.III$	3281.05	16196
$R_2.III$	2168.37	14916	$AR_2.III$	3016.02	14916
$R_3.III$	2933.3	20092	$AR_3.III$	4078.9	20092
$R_4.III$	8658.63	57095	$AR_4.III$	12016.9	57095
$R_5.III$	21431.76	252628	$AR_5.III$	94717.91	392638
$R_6.III$	13813.24	39181	$AR_6.III$	14703.03	39181
$R_7.III$	17031.08	49314	$AR_7.III$	18312.63	49314
$R_8.III$	39453.53	111077	$AR_8.III$	42234.65	111077
$R_9.III$	110781.39	286352	$AR_9.III$	117406.11	286352
$R_{10}.III$	182693.16	2375496	$AR_{10}.III$	186618.16	4135482
$R_{11}.III$	25890.79	62447	$AR_{11}.III$	27859.34	62447
$R_{12}.III$	49063.62	116787	$AR_{12}.III$	52703.77	116787
$R_{13}.III$	73563.22	170375	$AR_{13}.III$	78953.81	170375
$R_{14}.III$	166110.05	382111	$AR_{14}.III$	333792.74	661431
$R_{15}.III$	145279.9	3840277	$AR_{15}.III$	265010.09	10341540
$R_1.IV$	2808.9	18541	$R_1.IV$	4726.67	23541
$R_2.IV$	3837.71	26159	$AR_2.IV$	4344.63	21681
$R_3.IV$	6973.63	46613	$AR_3.IV$	5875.02	29204
$R_4.IV$	18420.13	115993	$AR_4.IV$	21514.43	102953
$R_5.IV$	51775.43	313499	$AR_5.IV$	136442.69	570714
$R_6.IV$	19516.48	80196	$AR_6.IV$	21180.31	56951
$R_7.IV$	22308.32	94986	$AR_7.IV$	26379.77	71680
$R_8.IV$	49303.2	282283	$AR_8.IV$	60839.19	161455
$R_9.IV$	138439.96	355348	$AR_9.IV$	169126.67	416223
$R_{10}.IV$	123138.68	2947867	$AR_{10}.IV$	217712.76	6011076
$R_{11}.IV$	32354.48	104441	$AR_{11}.IV$	40132.12	90769
$R_{12}.IV$	61312.76	264009	$AR_{12}.IV$	75919.83	169754
$R_{13}.IV$	142360.74	466005	$AR_{13}.IV$	113733.68	247646
$R_{14}.IV$	321457.65	743334	$AR_{14}.IV$	480835.53	961414
$R_{15}.IV$	280651.2	7470630	$AR_{15}.IV$	736773.45	15031811

Table A.11: Computational Results obtained with the $\mathcal{RLA}$ on the SpRand and SpAcyc networks.

	$\mathcal{FLSM}.\mathcal{LS}.2^{nd}$		$\mathcal{BDLSM}.\mathcal{LS}.2^{nd}$	
	CPU	# Iter.	CPU	# Iter.
*c*101	5460.05	32674.75	9679.47	11663.61
*c*102	309146.08	478896.86	145755.35	96187.22
*c*103	OoM	OoM	382925.23	195931.81
*c*104	OoM	OoM	OoM	OoM
*c*105	33648.1	94287.8	31001.15	28907.62
*c*106	29129.85	123739.07	46001.04	50838.25
*c*107	52428.12	220173.65	68367	63260.7
*c*108	92783.15	311397.05	113612.74	103431.48
*c*109	OoM	OoM	305340.95	213659.72
*r*101	156.64	515.15	421.47	559.57
*r*102	8455.55	22851.55	15423.74	7597.23
*r*103	11622.44	47300.09	24978.1	19758.9
*r*104	28231.73	95857.26	46537.32	30136.8
*r*105	952.04	7491.93	2657.57	4221.89
*r*106	6583.13	36850.3	17803.82	19639.21
*r*107	14992.43	60743.4	36893.64	31808.55
*r*108	26482.12	91481.4	62811.77	49833.94
*r*109	3837.81	25057.76	12778.19	18576.96
*r*110	8923.16	45480.46	24961.34	27107.28
*r*111	10359.26	49042.03	29707.94	37699.89
*r*112	13541.28	55778.46	35331.74	33416.56
*rc*101	1997.61	9139.39	4453.5	7243.95
*rc*102	13541.36	34840.77	18362.14	20625.86
*rc*103	16879.79	69133.27	43455.47	34090.49
*rc*104	30279.82	101443.22	69654.97	50543.55
*rc*105	2979.09	20556.18	8534.16	10404.88
*rc*106	3697.9	24511.32	16834.66	25995.53
*rc*107	9282.86	48633.74	24988.39	25791.3
*rc*108	14898.69	65198.16	36419.68	33466.73

Table A.12: Computational Results obtained with the label selection methods on the Solomon's Networks OoM indicates out of memory.

	$\mathcal{FNSM}.\mathcal{SALS}.2^{nd}$		$\mathcal{BDNSM}.\mathcal{SALS}.2^{nd}$	
	CPU	# Iter.	CPU	# Iter.
*c*101	874.73	546.64	453.7	219.39
*c*102	257712.37	942.93	10187.82	495.94
*c*103	OoM	OoM	34820.78	684.99
*c*104	OoM	OoM	85348.74	998.18
*c*105	11481.98	725.16	1685.88	352.73
*c*106	7113.6	747.54	3401	376.94
*c*107	7628.86	750.69	3884.27	416.81
*c*108	19624.35	903.63	9328.59	666.59
*c*109	77828.58	1233.04	28611.45	838.35
*r*101	110.35	166.97	351.75	227.13
*r*102	4774.02	571.99	3518.54	550.38
*r*103	16193.56	674.52	3650.39	622.27
*r*104	12917.17	848.8	9298.71	773.63
*r*105	312.65	484.8	515.31	443.44
*r*106	2995.03	697.52	2512.88	639.71
*r*107	7457.73	877.99	5834.43	789.47
*r*108	14352.89	1017.72	12059.26	830.79
*r*109	1591.24	632.29	2121.13	635.98
*r*110	3947.38	817.79	4758.79	838.76
*r*111	4727.87	757.23	5085.84	719.78
*r*112	8237.64	1212.33	8845.71	1056.62
*rc*101	312.83	442.48	546.69	431.74
*rc*102	2979.82	626.97	2902.37	644.36
*rc*103	8050.04	912.25	7082.35	939.75
*rc*104	15554.45	1059.13	13494.43	1026.39
*rc*105	1139.22	517.61	1435.79	484.18
*rc*106	1732.32	799.75	2403.67	775.86
*rc*107	4165.86	818	5008.6	711.6
*rc*108	7551.04	1233.37	8377.76	924.09

Table A.13: Computational Results obtained with the node selection methods on the Solomon's Networks OoM indicates out of memory.

Appendix B - Algorithms illustrations

In what follows, we illustrate the operations executed by the proposed node selection and label selection approaches during the first iteration, by considering the network reported in Figure B.1. It is assumed that the origin node is $s = 1$ and the destination node is $d = 7$.

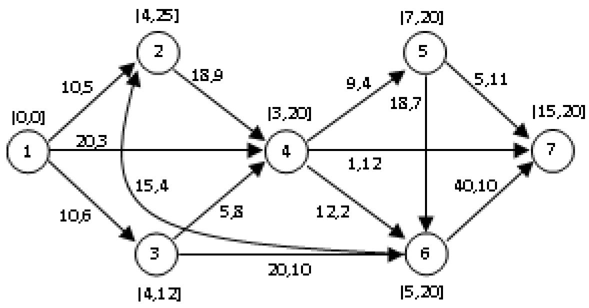

Figure B.1: Problem data. Time window shown next to each node. Cost/time shown next to each arc.

For each pair of nodes i and j, Table B.1 gives the value of the lower bound LB_{ij}, used to identify the unreachable nodes.

From	To						
	1	2	3	4	5	6	7
1	-	5	6	3	7	5	15
2	-	-	-	9	13	11	21
3	-	14	-	8	12	10	20
4	-	6	-	-	4	2	12
5	-	11	-	20	-	7	11
6	-	4	-	13	17	-	10

Table B.1: Lower bound values for each pair of nodes - indicates that a path connecting the considered nodes does not exist.

In Figures B.2-B.7 the labels, after the first iteration of the proposed approaches, are shown next to the nodes. They take the following form $[(S), C, T, t]$ where S is the vector of dummy node resources, C is the cost, T is the time and t is the departure time from the destination node; this last value is present only in the labels generated during the backward phase. In the bi-directional algorithms the labels reported in bold are those generated during the forward phase.

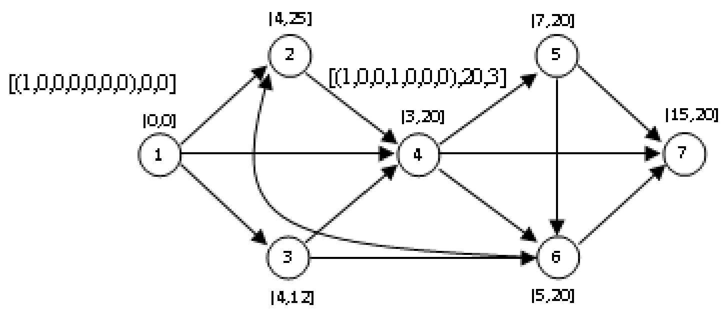

Figure B.2: At the initialisation phase the label at node 1 is created and it is added into set L, This label is selected and removed from L and the possibility to extend it to node 2, 3 and 4 is evaluated. The only generated label is that of node 4, while the potential labels for node 2 and 3 have node 7 as an unreachable node. For this reason they are not stored.

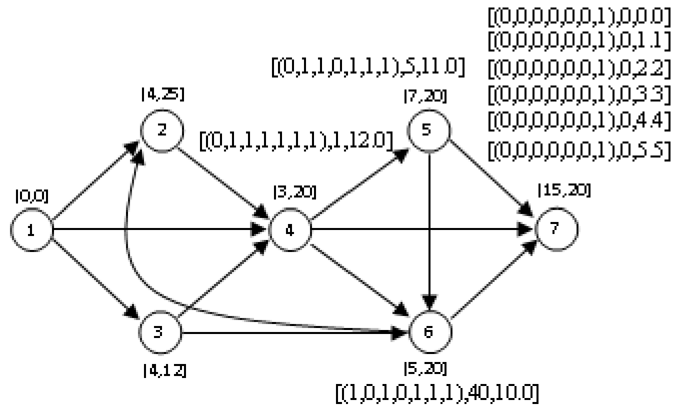

Figure B.3: In the $\mathcal{BLSM}$ six labels are created at node 7 each of them starting to different feasible instant time and they are included into set L. The first label, starting at instant zero (that has the possibility to represents a path from node 1 to node 7 with a value of T equals to 20), is selected and removed from L. One label to each node 4, 5 and 6 is generated and stored, respectively. All these new labels have the possibility to reach node 1.

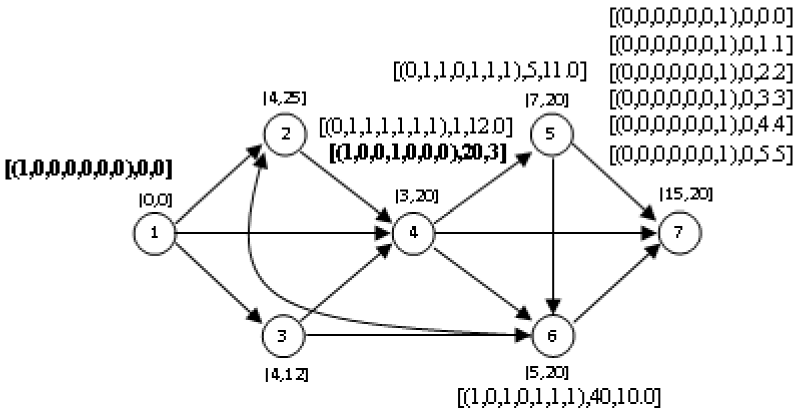

Figure B.4: One forward label and six backward labels are created at node 1 and 7, respectively. First, from the set L^{fw} the label is selected and a new label is generated and stored only for node 4. The motivation are that explained in Figure B.2. In the second phase, a label is selected from the top of L^{bw} and labels are generated for node 4, 5 and 6.

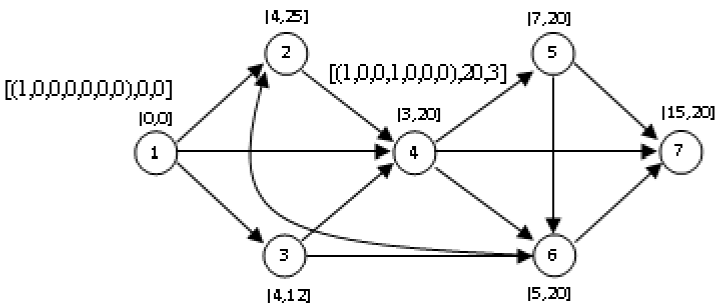

Figure B.5: In the $\mathcal{FNSM}$ a label for node 1 is created and the node is added into the list L. Node 1 is selected and removed from L. The label at node 1 is extended to node 2, 3 and 4. Only the label associated with node 4 is stored, because the labels for nodes 2 and 3, generated starting from the label associated with the selected node, do not have the potential to reach node 7 in its time window.

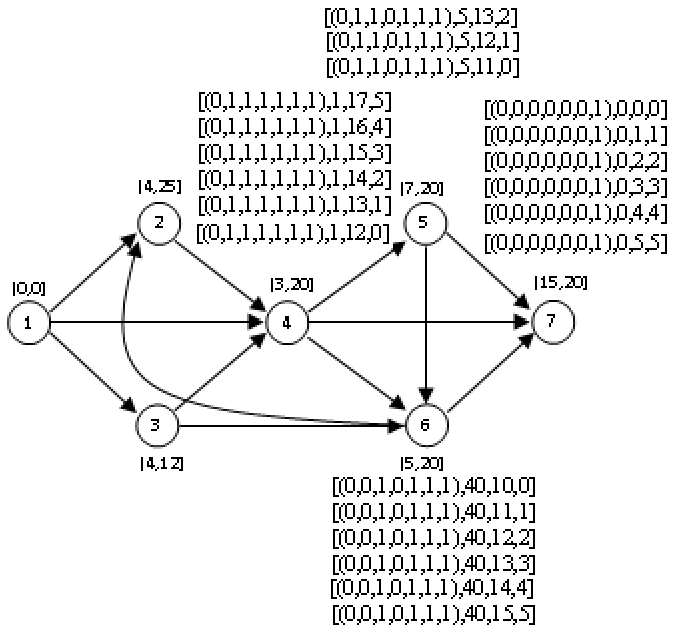

Figure B.6: Node 7 is added to the list L and for each feasible instant time, a label is created for node 7. This node is selected and removed from L. All the labels belonging to $D^{bw}(7)$ are extended to nodes 4, 5 and 6. The labels generated for nodes 4 and 6 are all stored, while for node 5 only the first three labels are considered, since the labels generated from those that start from node 7 at instant time 3, 4 and 5 have node 1 as an unreachable node.

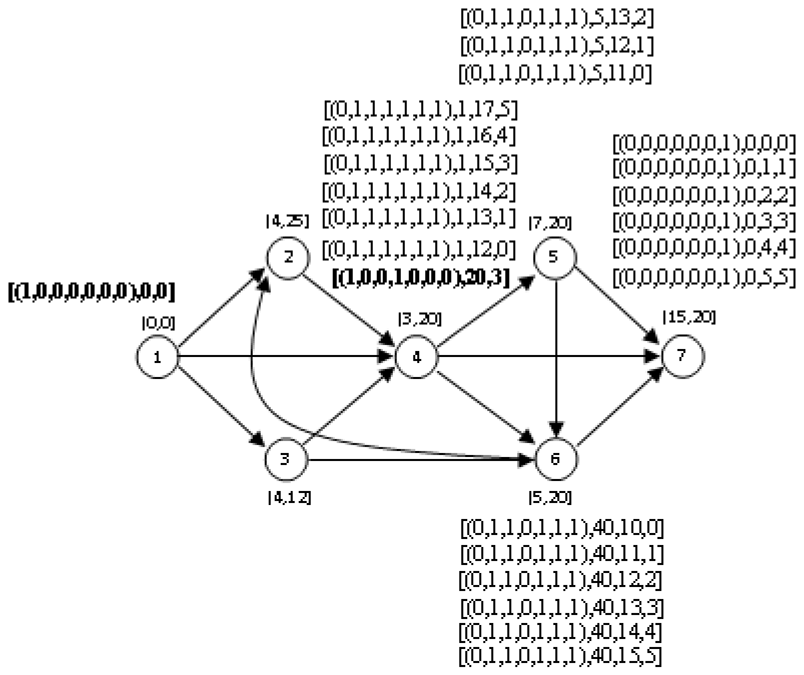

Figure B.7: One forward label and six backward labels are created at node 1 and 7, respectively. In the forward phase, from the set L^{fw} node 1 is selected and a new label is generated and stored only for node 4. In the backward phase, the node 7 is selected and removed from L^{bw}. The labels at node 7 are used to generate the sets of labels for node 4, 5 and 6, respectively. Also in this case for node 5 only three labels are stored.

Chapter 6

A computational study of the resolution methods for the resource constrained elementary shortest path problem

Luigi Di Puglia Pugliese

Department of Electronics, Informatics and Systems, University of Calabria, 87036, Rende (CS), Italy

Francesca Guerriero

Department of Electronics, Informatics and Systems, University of Calabria, 87036, Rende (CS), Italy

Abstract

We present the results of a computational investigation of solution approaches for the resource constrained elementary shortest path problem ($\mathcal{RCESPP}$). In particular, the best known algorithms were tested on several problem instances taken from the literature. The main aims are to provide a detailed state of the art and to evaluate methods that turn out to be the most promising for solving the problem. This work represents the first attempt to computationally compare solution approaches for the $\mathcal{RCESPP}$.

Keywords: resource constraints; elementary shortest path; dynamic programming.

6.1 Introduction

The Resource Constrained Elementary Shortest Path Problem ($\mathcal{RCESPP}$) is to find the least cost path from a source node to a destination node, for which the resources consumed along this path must satisfy feasibility criteria.

The $\mathcal{RCESPP}$ arises when the Vehicle Routing Problem ($\mathcal{VRP}$), with additional constraints, is solved via column generation and branch and price methods. In practice, the additional constraints to the classical $\mathcal{VRP}$ are related to the capacity of the vehicles and to the specific requirements of the customers/nodes. The latter case usually refers to the service time, i.e. a time window is associated with each castomer/node. It is important to point out that both soft and hard time windows constraints can be considered. In the former case, the visiting of a node is allowed also out the time window and a penalty cost is paied (see the work of Qureshi et al. [117]). In the latter case, the service time must fall within the time windows. Refering to the aforementioned constraints, in the scientific literature three main instances of the constrained $\mathcal{VRP}$ were defined: the Capacitad $\mathcal{VRP}$ (shortly $\mathcal{CVRP}$, see Toth and Vigo [133]); the $\mathcal{VRP}$ with Distribution and Collection (shortly $\mathcal{VRPDC}$, see Dell'Amico et al. [48]); the Capacited $\mathcal{VRP}$ with Time Windows (shortly $\mathcal{CVRPTW}$, see Cordeau et al. [37]). When the aforementioned problems are solved via branch and price method, the pricing problem requires to find a $\mathcal{RCESPP}$ on a graph with positive cost on the arcs and non-negative prizes on the nodes. The cost of the path is given by the sum of the cost on the arcs traversed minus the sum of the prizes on the visited nodes. The prizes can be taken into account considering the cost on the arcs computed by subtracting from the original cost the prize on the tail node. It is evident that in this case the cost of the path is only the sum of the cost on the arcs that belong to the path. In addition, since the prizes are non-negative values, the cost on the arcs can be negative.

Another class of problems that is solved via column generation method is represented by the short-haul aircraft rotation problems ([13]). Also in this case, the pricing problem is mathematically formulated as the $\mathcal{RCESPP}$.

Since the $\mathcal{RCESPP}$ is defined on general graphs and the costs are not constrained in sign, it is necessary to impose that the optimal solution must be elementary, that is a node cannot appear more than once in the optimal path. The $\mathcal{RCESPP}$ has been proven to be strongly $\mathcal{NP}$-hard by Dror in [59].

The scientific literature provides several optimal solution approaches. Tipically, the proposed methods are based on the dynamic programming formulation of the problem. The main differences concern the way in which the constrain related to the elementarity of the optimal path is taken into account.

To the best of our knowledge, the most efficient solution approaches to solve the $\mathcal{RCESPP}$ are those based on the idea of Kohl [96]. The scientific literature provides two algorithms based on this concept, that is the solution approach proposed by Boland et al. [22] and the method of Righini and Salani [120]. It is important to point out that the two algorithms have been tested on different sets of instances and they have been developed and published in the same time period. The main aim of this work is to compare the two algorithms on the same set of test problems and to provide a comprehensive computational study to prove the practical behaviours of both methods. In addition, the considered algorithms can be viewed as derived from a unique framework. In this paper, we formalize the prototype method from which the solution approaches studied in this work can be derived.

The outline of this paper is given as follows: in the next Section we review the solution approaches proposed in the scientific literature to address the $\mathcal{RCESPP}$. In Section 6.3 we give the formal definition of the $\mathcal{RCESPP}$. The prototype framework is presented in Section 6.4. A description of the method proposed in [120] is given in Section 6.5 while in Section 6.6 we describe in detail the solution approach developed in [22]. Comments about the efficiency of the methods are given in Section 6.7. Section 6.8 is devoted to the empirical comparison of the performance of the considered solution approaches. Final remarks and conclusions are given in Section 6.9.

6.2 State of the art

In the last ten years, the $\mathcal{RCESPP}$ have been well studied. Several optimal solution approaches based on dynamic programming formulation have been developed starting to the seminal work of Beasley and Christofides ([16]). Before this work, several papers addressed the problem by solving a relaxation, that is the Resource Constrained Shortest Path Problem ($\mathcal{RCSPP}$), with the aim to eliminate only cycles of length two. All these works used the procedure of eliminating cycles of length two proposed in the context of the shortest path problem, that was addressed first by Houck et al. [86] and Christofides et al. [32].

A wide number of papers have tackled the $\mathcal{RCSPP}$ and several efficient algorithms have been proposed. Briefly, we can classified these approaches in three main groups: 1) solution approaches based on the k-shortest path ([125], [81], [105]); 2) those based on dynamic formulation ([94], [97], [53], [52], [50], [92], [60]); 3) methods that close the duality gap obtained with the Lagrangian relaxation (Handler and Zang [81], Beasley and Christofides [16]). Works that appear in more than one group propose solution approaches that combine different methods.

In the work of Beasley and Christofides [16], for the first time the idea to optimality solve the $\mathcal{RCESPP}$ via dynamic programming procedures was introduced. The authors gave an integer programming formulation for the $\mathcal{RCESPP}$ considering either lower and upper limits on the consumption of resources. In addition, no arcs can appear more than once in the optimal solution. The authors used the same formulation to model also the non elementary counterpart. However, they pointed out that the latter formulation allows to reach an elementary path if the graph does not contains negative cost cycles and the lower limits on the consumption of resources equals zero. On the other hand, it is necessary to impose that the solution is elementary. Beasley and Christofides discussed on the possibility to define a dynamic programming recursion that is able to solve both models. It is important to point out that if either the graph contains negative cost cycles or the lower limits are not trivial, then the solution found cannot be elementary. To apply dynamic programming procedure for optimally solving the $\mathcal{RCESPP}$, the authors proposed to consider further resources, one for each node and the lower and upper limits are set to zero and one, respectively. In other words, the $\mathcal{RCESPP}$ is treated as the $\mathcal{RCSPP}$ with additional (node) resources. Since the state-space increases drastically with respect to the number of resources, the same authors underlined that the dynamic programming recursion could be not efficient, thus they discarded this formulation for the $\mathcal{RCESPP}$ and did not conduct any computational tests.

The idea of associating extra resources with each node was later studied by Kohl in [96]. The author proposed a solution approach that repeatedly solves a relaxation of the state-space of the dynamic programming algorithm related to the $\mathcal{RCSPP}$ with the additional node resources introduced by Beasley and Christofides in [16]. Kohl suggested to introduce additional resources only for some nodes. This yields a relaxation of the $\mathcal{RCESPP}$, which could be solved to obtain an optimal path which may contain repeated nodes. An extra resource is added for each node that appears more than once in the optimal solution obtained so far. The problem defined on the new state-space is re-solved, until the optimal solution does not contain cycles. In addition, the author proposed an acceleration of this method. In particular, he observed that two or more nodes cannot be part of the same path becouse of the resource limitation. In this case, only one resource is introduced that is associated with this set of nodes. In other words, if a node of this set is part of a sub-path, the consumption of the additional resource is set equal to one and no other node belongs to the set can be reached from this sub-path. As in the case of Beasley and Christofides, the idea was not implemented and no numerical experiments were conducted.

Fifteen years later the work of Beasley and Christofides ([16]), Feillet et al. in [64] gave computational studies related to the idea of additional node resources as proposed in [16]. The authors extended the label correcting algorithms proposed in [50] associating with each node extended labels. In particular, each label is characterized by the cost and the resources consumption as in [50] and by n binary extra resources. It is important to point out that a label associated with node i corresponds to a path from node s to node i. The consumption of the additional resources is set equal to zero and the corresponding label is updated when the node is visited. In other words, the additional resources have a consumption equal to one for each node that belongs to the path associated with the label. On the other hand, the availability of such resources equals zero. Feillet et al. extended the concept of dominance relation on the newly defined label. In particular, a label cannot be dominated by one that has a higher number of visited nodes. With these considerations, the label correcting algorithm proposed by Desrochers ([50]) can be trivially extended, that is the new definition of label ensures that all efficient solutions related to elementary paths are found, thus the minimun feasible path from the source node to the destination node is obtained as well. The same authors underlined that the size of the state-space increases drastically, and one is likely to generate and to keep many more labels during the algorithm. With the aim of reducing the number of generated labels, the authors introduced the concept of unreachable nodes. For a given label, the binary additional resources are set to one also for the nodes that cannot be reached from the sub-path associated with the label, because of resource limitations. With this modification, the dominance relation becomes sharper and it is more likely to fathom unpromising sub-paths. It is important to point out that the concept of unreachable nodes introduced in [64] resembles the acceleration idea of Kohl ([96]). The difference is that Kohl used this concept to reduce the state-space while in Feillet et al. ([64]) the unreachable nodes are introduced to accelerate the method from the point of view of the running time.

For the first time Boland et al. in [22] implemented and numerically tested methods based on the idea of Kohl ([96]). The authors defined an algorithm called General State-Space Augmenting Algorithm ($\mathcal{GSSAA}$, for short) that repeatedly solves a $\mathcal{RCSPP}$ with extra node resources using a general label setting algorithm ($\mathcal{GLSA}$, for short). The $\mathcal{GSSAA}$ terminated when the $\mathcal{GLSA}$ returns an elementary path. On the other hand, further (node) resources are introduced. The authors proposed many strategies to increase the resources at each step of the $\mathcal{GSSAA}$. These strategies differ in how the repeated nodes in the optimal solution of the $\mathcal{GLSA}$ are chosen for adding the extra resource. Boland et al. [[22]] used the concept of dominance and unreachable nodes introduced by Feillet et al. in [64] to solve the relaxed $\mathcal{RCSPP}$ with additional resources. The authors tested their methods on randomly generated networks after a preprocessing stage. In particular, their preprocessing is divided in two main phases. In the first phase, the procedure of Aneja et al. ([5]) is runned with the aim of reducing the size of the networks, in the second phase the least resource path for each resource and for each pair of node is computed.

The output of the second phase is used to identify the unreachable nodes.

Righini and Salani in [119] proposed an enhanced version of the dynamic programming algorithm proposed by Feillet et al. ([64]). Their method takes advantages by the bi-directional seach. In particular, the label are extended both in forward and in backward starting from the source node and the destination node, respectively. When there are no more labels to be scanned, the optimal path is determined by joining forward and backward paths. The authors proposed bounding criteria with the aim to stop the extention of a label in one direction when it is guarantee that the remaining part of the path will be generated in the other direction. The authors tested their method on Solomon's instances and they concluded that the use of a bounding bi-directional search outperforms the mono-directional counterpart. Two year later the same authors in [120] applied the bi-directional search strategy to solve a relaxation of the $\mathcal{RCESPP}$. In their work they considered three different solution approaches: exact dynamic programming, branch-and-bound based on state-space relaxation and decremental state-space relaxation ($\mathcal{DSSR}$, for short). The first approach is the same proposed in [119], in the second case lower bounds, that is the optimal solutions of the relaxed $\mathcal{RCESPP}$, computed with the dynamic programming that uses the bounding bi-directional search are exploited in a branch-and-bound framework. The $\mathcal{DSSR}$ works as the $\mathcal{GSSAA}$ proposed by Boland et al. in [22]. The main difference concerns the dynamic programming algorithm used to solve iteratively the relaxed problem. The authors concluded that the last approach is faster than both exact dynamic programming and the branch-and-bound algorithms. Also in this case the tests were conducted on complete networks taken from the benchmark instances of Solomon.

It is worth observing that Righini and Salani make use of the $\mathcal{DSSR}$ to solve also the orienteering problem with time windows

([121]). In their work, the authors consider the strategies proposed by Boland et al. in [22] to increment the set of critical nodes. The computational results underlined that no strategy dominates the others on the Solomon's banchmark instances. In addition, the authors introduce ad hoc heuristics for the orienteering problem with the aim of a priory identify some critical nodes in order to initialize the related set. They conclude that the initialization of the set of critical nodes rather than an initial empty set, accelerate the seach of the optimal solution.

6.3 Problem definition and notation

The $\mathcal{RCESPP}$ is defined on a graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$, where $\mathcal{N}$ is the set of n nodes and $\mathcal{A}$ represents the set of $m \leq |\mathcal{N} \times \mathcal{N}|$ arcs. With each arc $(i, j) \in \mathcal{A}$ is associated a cost c_{ij} and a resource vector $w_{ij} \in \mathcal{R}_{+}^{R}$. In particular, c_{ij} represents the cost to traverse the arc $(i, j) \in \mathcal{A}$, while w_{ij}^{r} is the consumption of resource $r, r = 1, \ldots, R$ along the arc $(i, j) \in \mathcal{A}$. It is important to point out that resources consumed to the nodes instead to the arcs can be easily considered as arcs resources. In particular, it is sufficient to set the resources consumption of each ingoing arc of a given node i equal to the resources consumption associated with i.

Given two distinct nodes s (referred to as source node) and d (referred to as destination node), a *path* from s to d is a sequence of nodes $\Pi_{sd} = \{s = i_1, \ldots, i_l = d\}$, $l \geq 2$ and a corresponding sequence of $l - 1$ arcs such that the k-th arc in the sequence is $(i_k, i_{k+1}) \in A$ for $k = 1, \ldots, l - 1$. Thus, each path contains at least one arc. A path is said to be *elementary* if it does not contain repeated nodes. An *oriented cycle* is an elementary path for which the source and destination nodes are the same.

Since, $\forall (i, j) \in \mathcal{A}$ the cost c_{ij} is not constrained in sign the graph can contain negative cost cycles. The aim is to find the least cost path from s to d, considering constraints on the resources consumption. In particular, for each node i and for each resource $r = 1, \dots, R$ upper limits $\overline{W_i^r}$ and lower limits $\underline{W_i^r}$ are given, that is the resources consumed along a path from s to i must be less than or equal to the upper limit $\overline{W_i^r}$ and must be greather than or equal to the lower limit $\underline{W_i^r}$. In addition, a feasible path cannot contain repeated nodes that is the path must be elementary.

Let Π_{sd} be a path from node s to node d. We define with $N(\Pi_{sd}) = \{i_0 = s, i_1, \dots, i_l = d\}$, $l \geq 1$ and $A(\Pi_{sd}) = \{(i_v, i_{v+1}), v = 0, \dots, l-1\}$ the set of nodes and the set of arcs that belong to Π_{sd}, respectively. The cost of the path Π is $c(\Pi) = \sum_{(i,j) \in A(\Pi)} c_{ij}$ and the resource consumed along Π is $w(\Pi) = \{w^1(\Pi), \dots, w^R(\Pi)\}$. Depending on the nature of the resource, the value $w^r(\Pi)$ of each resource $r = 1, \dots, R$ for the given path Π can be defined in different way. The optimal solution is the feasible path with minimum cost.

The $\mathcal{RCESPP}$ can be mathematically formulated as follows.

$$\min\nolimits_{\Pi_{sd} \in \bar{\Pi}} c(\Pi_{sd}) \tag{6.1}$$

$$s.t.$$

$$\underline{W_i^r} \leq w^r(\Pi_{si}) \leq \overline{W_i^r}, \ \forall r = 1, \dots, R, \tag{6.2}$$
$$i \in N - \{s\}, \ \Pi_{si} \in \bar{\Pi},$$

where $\bar{\Pi}$ is the set of all elemetary paths in $\mathcal{G}$ from node s to each other node $i \in N - \{s\}$.

The general formulation of the problem reported above takes into account both time windows and capacity constraints. In the first case, the lower limit of node i represents the earliest arrival time to the node, that is a_i while the latest arrival time to node i, that is b_i is viewed as the upper limit $\overline{W_i^r}$. In the case of capacity constraints, the lower limits $\underline{W_i^r}$ is set equal to zero, while the upper limit equals the capacity of the vehicle.

When the $\mathcal{RCESPP}$ is solved by a dynamic programming procedure, H_i labels $y_i^h = (S_i^h, W_i^h, C_i^h)$, $h = 1, \ldots, H_i$ are associated with each node $i \in \mathcal{N}$. Each label y_i^h contains the characteristics of the $h - st$ path Π_{si}^h from node s to node i, that is $W_i^h = w(\Pi_{si}^h)$ and $C_i^h = c(\Pi_{si}^h)$. The vector $S_i^h \in \mathcal{R}^n$, stores the consumption of resource at node u_p, $p = 1, \ldots, n$. In particular, for each node u_p, if $u_p \in N(\Pi_{si}^h)$ then $S_i^h[p] = 1$, otherwise $S_i^h[p] = 0$. In a dynamic programming algorithm, labels are extended to generate feasible labels. This recursion is run until no more labels can be extended. The label with the minimum cost associated with the destination node d is the optimal solution. Different Resource Extension Functions ($\mathcal{REF}s$) can be defined, depending on the nature of the resources. For a detailed description and classification of $\mathcal{REF}s$ the reader is refered to the work of Irnich ([89]).

Since solving the $\mathcal{RCESPP}$ means to find a feasible path with minimum cost and since with each path is associated a label, in what follows the terms "solution", "path" and "label" are used in unterchangeable fashion.

Definition 6.3.1. *Let y_i^1 and y_i^2 be two labels associated with node i. We say that label y_i^1 dominates y_i^2 if $S_i^1 \leq S_i^2$, $W_i^1 \leq W_i^2$, $C_i^1 \leq C_i^2$ and at least one inequality is strictly.*

Definition 6.3.2. *A solution is said to be efficient if other solutions, that dominate it, do not exist.*

Definition 6.3.3. *A path Π_{si} from node s to node i is said to be feasible if $\underline{W_u} \leq W_u \leq \overline{W_u}$, for all $u \in N(\Pi_{si})$.*

Definition 6.3.4. *Let $M_\Pi(j)$ be the multiplicity of node j in path Π, that is $M_\Pi(j) = |\{v : 0 \leq v \leq |N(\Pi)|, i_v = j\}|$. The path Π is said to be an elementary path if $M_\Pi(j) = 1$, for all $j \in N(\Pi)$.*

We denote with $D(i)$ the set of all efficient solutions associated with node i and with $FS(i)$ and $BS(i)$ the successor nodes and the predecessor nodes of node i, respectively. In formal terms, we have that $FS(i) = \{j : (i,j) \in \mathcal{A}\}$ and $BS(i) = \{u : (u,i) \in \mathcal{A}\}$.

6.4 Prototype framework

Following the idea of Kohl ([96]), a dynamic programming algorithm is used to solve the relaxed $\mathcal{RCSPP}$ with additional node resources, that is a resource is considered only for a set of nodes that represent the so-called *critical nodes*. When the dynamic programming algorithm is executed, multiple visit at the critical nodes is avoided, that is $M_{\Pi^h_{si}}(j) \leq 1$, $\forall i \in N, h = 1, \ldots, H_i$ and $j =$ critical node, whereas the other nodes can appear more than once in the path. This behaviour is obtained replacing the binary vector S with a binary vector $\hat{S}$. It is evident that the size of $\hat{S}$ is restricted only to the critical nodes.

Let $\Pi^*_{\hat{S}}$ be the optimal solution of the $\mathcal{RCSPP}$ with additional node resources restricted to $\hat{S}$. If $\Pi^*_{\hat{S}}$ is not an elementary path, then the repeated nodes of the optimal path are marked as critical nodes and the dynamic programming algorithm is applied again. The procedure is iterated until the optimal solution of the relaxed problem is an elementary path. Let $\mathcal{P}^*$ be the set of path Π^h_{sd} associated with all efficient labels $y^h_d \in D(d)$. The idea of Incremental Node Resources ($\mathcal{INR}$ for short) proposed by Kohl is formalized in Algorithm 17.

It is important to point out that both the $\mathcal{GSSAA}$ proposed by Boland et al. [22] and the $\mathcal{DSSR}$ presented by Righini and Salani [120] can be derived from the prototype $\mathcal{INR}$ framework. The main differences are related to the **Step 1** and to the way in which the set $\hat{S}$ is updated at each iteration. These issues are better explained is Section 6.7.

For the sake of completeness, in the next Sections the $\mathcal{DSSR}$ and the $\mathcal{GSSAA}$ algorithm are described in details.

Algorithm 17 $\mathcal{INR}$ framework

```
Step 0 (Initialization)
Set: $\hat{S} = \emptyset$; $\Pi^*_{\hat{S}} = \emptyset$.

Step 1 (Relaxed problem solution)
Solve the $\mathcal{RCSPP}$ with resources node restricted to $\hat{S}$ obtaining $\mathcal{P}^*$.

Step 2 (Optimality check)
if $\Pi^*_{\hat{S}}$ is not an elementary path then
    Update set $\hat{S}$,
    go to Step 1.
else
    STOP,
    $\Pi^*_{\hat{S}}$ is the optimal solution for $\mathcal{RCESPP}$.
end if
```

6.5 The decremental state-space relaxation

The $\mathcal{DSSR}$, proposed by Righini and Salani in [120], can be viewed as a compromise between the exact dynamic programming approach, where $S \in \mathcal{R}^n$ and the state-space relaxation. The former approach resemble the idea of Feillet et al ([64]), while the latter has been proposed by Righini and Salani in [120]. In particular, a label correcting method is used to find the set of feasible and efficient solutions from node s to node d, by considering label (σ, W, C) instead of (S, W, C), where $\sigma = \sum_{p=1,\dots,|N|} S[p]$ represents the length of the path associated with the label, that is the number of nodes visited, excluding s. It is evident that since σ does not keep information about the set of already visited nodes, cycles are not forbidden. As a conseguence, it is ensured that the path is feasible but it could be not elementary.

When all nodes are critical, the $\mathcal{DSSR}$ is equivalent to the exact dynamic programming; when $\hat{S}$ is empty it is equivalent to the algorithm with state-space relaxation.

The $\mathcal{DSSR}$ iteratively solves the $\mathcal{RCSPP}$ with a specific set of critical nodes. If $\Pi^*_{\hat{S}}$ contains cycles, then each node $j \in \Pi^*_{\hat{S}}$ such that

$M_{\Pi^*_{\hat{S}}}(j) \geq 2$ is marked as critical node and the $\mathcal{RCSPP}$ is resolved. The algorithm terminates when $\Pi^*_{\hat{S}}$ is an elementary path.

The authors proposed a modified version of the label correcting algorithm introduced in [119], that makes use of a bounded bi-directional seach to iteratively solve the $\mathcal{RCSPP}$, in the proposed $\mathcal{DSSR}$. Their method is characterized by the execution of three main steps: the first step extends the labels in forward manner, in the second step the labels are extended in backward and in the last step forward and backward labels are joined together with the aim of producing feasible paths from node s to node d. A list L stores all the nodes for which at least one label associated with it has the potential to generate feasible paths. At each iteration, a node i is selected from the list L and all the labels that are not yet extended are used to generate new labels for each successor node in the first step while in the second step labels are extended to the predecessor nodes of node i.

For the sake of completness, in what follows we report the idea of bounded bi-directional seach proposed by Righini and Salani in [119].

The main idea is to extend labels from node s to its successors and from node d to its predecessors. In addition, with the bounding procedure shorter paths should be generated and duplicated path should be avoided. Both forward and backward labels are associated with each node. To store these two types of labels, the set $D(i)$ is divided into two distinct sets, that is $D^{fw}(i)$ and $D^{bw}(i)$, respectively. Each label $y_i^h \in D^{fw}(i)$, $h = 1, \ldots, H_i$ represents a path from node s to node i, while labels $y_i^k \in D^{bw}(i)$, $k = 1, \ldots, K_i$ are associated with paths from node d to node i. In [119] two bounding strategies are developed. The first considers lower bound on the number of arcs that can be part of a feasible solution, the second strategy is based on the resources consumption. The latter is more efficient than the former. The bounding strategy based on resources consumption is used to stop the extension of a label (both forward and backward) if the consumption of a critical resource $\hat{r}$, whose consumption is monotone along the paths, is greater than half of the upper limit $\overline{W_d^{\hat{r}}}$ associated with node d .

The $\mathcal{DSSR}$ iteratively uses the bounding bi-directional search to solve a relaxation of the $\mathcal{RCESPP}$, in which the condition related to the elementary path is considered only for the set of current critical nodes. At each iteration of the $\mathcal{DSSR}$, the current set of critical nodes, that is $\hat{S}$, is updated. In particular, the set Ψ is used to increment $\hat{S}$. In other words, to the set $\hat{S}$ are added the nodes that belong to the Ψ. The former set is composed by the nodes that are present more than once in the current optimal path. In formal terms, $\Psi = \{j : M_{\Pi^*_{\hat{S}}}(j) \geq 2\}$, thus $\hat{S} = \hat{S} \cup \Psi$.

In Algorithm 18, we report the pseudo-code of the bounded bi-directional search for a given set $\hat{S}$ proposed by Righini and Salani in [120] using the notation introduced in this work.

The function $Extend^{fw}(y_i, j)$ [or $Extend^{bw}(y_i, v)$] generates the label y_j [or y_v] by extending the label y_i along arc (i, j) [or (v, i)] following the $\mathcal{REF}$ for each resource $r = 1, \ldots, R$ and the consumption of the additional resource at node j [or node v] is set equal to one. To improve the performances of $\mathcal{DSSR}$, the authors used the technique to avoid cycles of length two proposed by Desrocher et al. in [51]. In addition, they make use of the concept of unreachable nodes proposed by Feillet et al. in [64]. In particular, given a label y_i, if for some $j \in \hat{S}$ and for some resource $r = 1, \ldots, R$ the condition $w^r(\Pi_{si}) + \underline{w^r_{ij}} > \overline{W^r_j}$ [or $w^r(\Pi_{id}) + \underline{w^r_{ji}} > \overline{W^r_j}$] holds, then the additional node resource is set equal to one, that is $\hat{S}_i[p] = 1$ such that $u_p \equiv j$. The value $\underline{w^r_{ij}}$, represents the least resource path from node i to node j, $\forall r = \overline{1, \ldots, R}$. It is important to point out that if for resource r the triangle inequality holds, then $\underline{w^r_{ij}} \equiv w^r_{ij}$.

Algorithm 18 Bi-directional search algorithm

```
Step 0 (Initialization)
Set: $y_s^1 = (\hat{S}_s^1, W_s^1, C_s^1)$ with $C_s^1 = 0$, $W_s^1 = (0, \ldots, 0)$, $D^{fw}(s) = y_s^1$, $D^{fw}(i) = \emptyset \quad \forall i \in N - \{s\}$;
Set: $y_d^1 = (\hat{S}_d^1, W_d^1, C_d^1)$ with $C_d^1 = 0$, $W_d^1 = (0, \ldots, 0)$, $D^{bw}(d) = y_d^1$, $D^{bw}(i) = \emptyset \quad \forall i \in N - \{d\}$
Set: $L = \{s, d\}$

Step 1 (Node selection)
Select a node $i$ from set $L$.

Step 2 (Forward extention)
for all $y_i^h = (\hat{S}_i^h, W_i^h, C_i^h) \in D^{fw}(i)$ do
  if $w^{\hat{r}}(\Pi_{si}^h) < \overline{W_d^{\hat{r}}}/2$ then
    for all $j \in FS(i)$ and $j \neq$ critical node or $j =$ critical node and $S_i^h[p] \neq 0$ for $p|u_p \equiv j$ do
      $y_j \leftarrow Extend^{fw}(y_i^h, j)$;
      if $y_j$ is not dominated by any label in $D^{fw}(j)$ then
        Set: $D^{fw}(j) = D^{fw}(j) \cup \{y_j\}$,
        remove from $D^{fw}(j)$ all labels that are dominated by $y_j$,
        add node $j$ to the list $L$ if it does not already belongs to it.
      end if
    end for
  end if
end for

Step 3 (Backward extention)
for all $y_i^k = (\hat{S}_i^k, W_i^k, C_i^k) \in D^{bw}(i)$ do
  if $w^{\hat{r}}(\Pi_{id}^k) < \overline{W_d^{\hat{r}}}/2$ then
    for all $v \in BS(i)$ and $v \neq$ critical node or $v =$ critical node and $S_i^k[p] \neq 0$ for $p|u_p \equiv v$ do
      $y_v \leftarrow Extend^{bw}(y_i^k, v)$;
      if $y_v$ is not dominated by any label in $D^{bw}(v)$ then
        Set: $D^{bw}(v) = D^{bw}(v) \cup \{y_v\}$,
        remove from $D^{bw}(v)$ all labels that are dominated by $y_v$,
        add node $v$ to the list $L$ if it does not already belongs to it.
      end if
    end for
  end if
end for

Step 4 (Termination check)
if $L = \emptyset$ then
  STOP. Go to Step 5
else
  Go to Step 1
end if

Step 5 (Join forward and backward paths)
Determine the set $\mathcal{P}^*$ by joining forward and backward paths.
return $\mathcal{P}^*$.
```

6.6 The general state-space augmenting algorithms

In this section, we provide a description of $\mathcal{GSSAA}$. Our description here is different from that provided in [22], although the algorithm is the same. This description uses terminology and concepts similar to $\mathcal{DSSR}$ to help clarify the similarities and differences between the two algorithms.

The basic idea of $\mathcal{GSSAA}$ is the same behind $\mathcal{DSSR}$. Also in $\mathcal{GSSAA}$, at each iteration the set of critical nodes is updated, that is $\hat{S} = \hat{S} \cup \Psi$. However, the set Ψ is defined in different ways. In particular, the authors proposed four different strategies to construct the set Ψ.

In the first case (HMO), Ψ contains only one node, that is the node with highest multiplicity in the feasible least cost path from node s to node d. The authors precised that if more than one least cost path exists, then the first one is selected. In addition, if more than one nodes has the multiplicity equal to the highest one, then the first of such node is putted in the set Ψ. The second strategy ($HMOAll$) uses all nodes with the highest multiplicity in the first feasible least cost path to construct set Ψ. When in the set Ψ are added all nodes with multiplicity greater than one in the first feasible least cost path, we refer to the third strategy ($MOAll$). In the last strategy, all feasible paths from node s to node d are considered. Thus, all nodes with a multiplicity greater than one are selected to be critical nodes (All).

A label setting method has been used to solve, at each iteration, the $\mathcal{RCSPP}$ with the extra resources associated with the critical nodes, that is with the resource vector $\hat{S}$. In particular, the algorithm of Desrochers et al. [51] has been modified to make use of the information obtained from preprocessing. Indeed, the authors utilized a preprocessing procedure that can be viewed as chacterized by the execution of two phases. In the first phase, the preprocessing strategy proposed by Aneja et al. [5] is performed with the aim of reducing the network size. In order to obtain lower bounds on the resources consumption, in the second phase of the preprocessing procedure, the shortest resource path for each pair of nodes and for each resource is computed. The amount of resource r consumed on the least resource path from node i to node j is defined as $\underline{w_{ij}^r}$, $\forall i, j \in N, i \neq j$ and $r = 1, \ldots, R$.

The performances of the label setting method were improved introducing the concept of strong dominance of Feillet et al. [64]. In other words, the resource consumption of node $j \in \hat{S}$ is set equal to one if node j cannot be reached from a generic path Π_{si}, that is node j is an unreachable node respect to the path Π_{si}. To identify this kind of node, the lower bounds obtained in the second phase of the preprocessing procedure are considered. Let y_i be a label that refers to a path Π_{si} from node s to node i. The resource consumption at node j is updated if $w^r(\Pi_{si}) + \underline{w^r_{ij}} > \overline{W^r_j}$ for some resource $r = 1, \ldots, R$. It is important to point out that these conditions generalize the conditions of unreachable node defined when the triangle inequality holds. In this case, the least resource path from node i to node j is the arc (i, j), thus $\underline{w^r_{ij}} \equiv w^r_{ij}$.

The steps of the $\mathcal{GLSA}$ to solve the $\mathcal{RCSPP}$ with node resources restricted to $\hat{S}$ are depicted in Algorithm 19. It is important to point out that the notations used here differ from those in the paper of Boland et al. [22]. However, the algorithm is the same.

6.7 Algorithm comparison

The algorithms briefly described in the previous Sections seem very similar, however several conceptual differences can be noted. The aim of this Section is to highlighted these aspects.

The similarity of $\mathcal{DSSR}$ and $\mathcal{GSSAA}$ is related to the idea of relaxing the state-space of the $\mathcal{RCSPP}$ with node resources. The studied algorithms represent the first attempt to implement and numerically test this idea that was theoretically studied by Kohl in [96]. However, we can identify four main aspects in which the two algorithms are substantially different: 1) the search procedure for finding the optimal state; 2) the extention strategy; 3) the definition of set Ψ; 4) the test problems setting.

Algorithm 19 General Label Setting Algorithm

```
Step 0 (Initialization)
Set: $y_s^1 = (\hat{S}_s^1, W_s^1, C_s^1)$ with $C_s^1 = 0$, $W_s^1 = (0, \ldots, 0)$, $D(s) = y_s^1$, $D(i) = \emptyset \quad \forall i \in N - \{s\}$;
Set: $L = \{y_i^1\}$

Step 1 (Label selection)
Select the label $y_i^h$ such that $W_i^h$ is lexicographically minimal.

Step 2 (Label extention)
for all $j \in FS(i)$ and $j \neq critical\ node$ or $j = critical\ node$ and $S_i^h[p] \neq 0$ for $p|u_p \equiv j$ do
    if $w^r(\Pi_{si}^h) + w_{ij}^r + \underline{w_{ij}^r} \leq \overline{W_j^r}$, $\forall r = 1, \ldots, R$ then
        $y_j \leftarrow Extend^{fw}(y_i^h, j)$;
        if $y_j$ is not dominated by any label in $D(j)$ then
            Set: $D(j) = D(j) \cup \{y_j\}$,
            remove from $D(j)$ and $L$ all labels that are dominated by $y_j$,
            add label $y_j$ to the list $L$.
        end if
    end if
end for

Step 3 (Termination check)
if $L = \emptyset$ then
    STOP,
    return $\mathcal{P}^*$.
else
    Go to Step 1.
end if
```

The search procedure of the dynamic programming based approaches, used to solve the relaxed state-space of the $\mathcal{RCSPP}$ with node resources, is the first main difference. The approach proposed by Boland et al. [22] is a classical label setting algorithm with slight modifications and the search procedure starts with an initial state associated with node s searching for the potential optimal states following a forward strategy.

The approach proposed by Righini and Salani in [120] is a label correcting method that resembles the algorithm of Feillet et al. [64]. The innovation is related to the search procedure. Indeed, the algorithm starts with node s and node d. At each iteration a node is selected and all labels associated with it are extended following both forward and backward strategies.

The second main difference is related to the selection strategy. In particular, the label setting algorithm of Boland et al. is based on a label selection strategy. In particular, the lexicographically minimal label is selected at each iteration to be treated. The selection strategy that is used in [120] is of node type. In particular, at each iteration the first node entering the queue is selected.

The third difference concerns the selection of the critical nodes. It is important to point out that Boland et al. [22] conducted a computational study considering several strategies to select at each iteration the critical nodes. We remark that four different strategies have been developed (see section 6.6). From their computational results, the authors concluded that the first strategy to compute Ψ, that is HMO, allows to achieve the best performances in terms of execution time. However, HMO executes the greatest number of iterations. This is coherent with the fact that a large number of critical resource allows to reduce the possibility of obtaining cycles in the path. Moreover, the performance of a dynamic programming algorithm, in terms of execution, gets worse when the number of resources increases. Righini and Salani selected the critical node following the procedure of Boland et al. [22], that we have classified as third strategy (see section 6.6).

Tha last difference, between the work of Righini and Salani [120] and the work of Boland et al. [22], is related to the computational experiments. In particular, in [22] random generated networks are considered, whereas, in [120] the benchmark Solomon's instances have been considered, In addition, Righini and Salani carried out computational test on three different instances of the $\mathcal{RCESPP}$: the elementary shortest path problem with capacity constraint ($\mathcal{ESPPC}$, for short), the elementary shortest path problem with distribution and collection ($\mathcal{ESPPDC}$, for short) and the elementary shortest path problem with capacity and time windows constraints ($\mathcal{ESPPCTW}$, for short).

It this paper we consider the $\mathcal{ESPPC}$ that is handled by both Righini and Salani ([120]) and Boland et al. ([22]).

6.8 Computational experiments

This section describes our experimental setup and testing methodology.

We have implemented the algorithm proposed by Boland et al. ([22]), that is the $\mathcal{GSSAA}$ and the $\mathcal{DSSR}$ defined in [120] by Righini and Salani. The considered solution strategies have been coded in Java and have been tested by using an intel(R) core(TM) i7 cpu M620, 2.67 GHz, ram 4.00 GB, under Microsoft 7 operating system. It is important to point out that the two algorithms have been implemented by using the same data structures, thus the comparison is not effected by implementation factors.

6.8.1 Test problems

In order to evaluate and to compare the $\mathcal{GSSAA}$ and the $\mathcal{DSSR}$, we have considered two sets of test problems.

The first set, that is S_1, contains instances derived from the Solomon's complete networks. The original data set proposed by Solomon is divided into random, clustered, and random-clustered categories, according to the displacement of the customers. Instances belonging to the same data-set have the customers located in the same way and with the same delivery requests; these instances differ only for the time windows.

The instances considered in this work are those proposed in [120], where the original Solomon's networks have been modified by associating a price with each vertex. As described in [120], this value is chosen randomly from the range $[0, 20]$. It is worth observing that the instances considered in this paper differ from those reported in [120] for the value of the price associated with each node. However, the structure of each network is the same. In particular, one instance taken from each one of the three Solomon's data-sets (namely c101, r101, and rc101) is considered. From each original instance, 10 $\mathcal{RCESPP}$ instances with 50 nodes and 10 $\mathcal{RCESPP}$ instances with 100 nodes have been derived, by choosing 10 different values for the vehicle capacity from 10 to 100. In the sequel, we refer to *test_nn_vc* to indicate the network *test* with *nn* nodes and a vehicle capacity equal to *vc*.

The second set of instances, that is S_2, is composed by fully random networks generated by using the public domain SPRAND generator ([29]). The dimension of the considered networks is equal to that considered in [22]. The characteristics of these problems are reported in Table 6.1. Starting from these networks, several instances have been generated by varying the percentage of negative arc cost. In the sequel, with the instance $R_1^{\%nc}$ we refer to the problem R_1 with a percentage of negative arc cost equal to $\%nc$. The resource limit has been computed as described in [22].

Test	Nodes	Arcs	Density
R_1	30	345	11,50
R_2	40	780	19,50
R_3	50	1000	20
R_4	50	1250	25
R_5	50	1500	30
R_6	50	2000	40
R_7	50	2250	45
R_8	50	2500	50
R_9	60	1770	30
R_{10}	100	4950	50
R_{11}	100	9900	99
R_{12}	150	16762	111,75

Table 6.1: Characteristic of the fully random networks.

It is important to point out that the preprocessing proposed in [22] is performed only for the second set of test problems. This means that the networks of set S_1 are not reduced and, since the triangle inequality holds, it is not necessary to compute the least resource path for each resource.

In addition, Righini and Salani in [121] have tested $\mathcal{DSSR}$ with the strategies proposed in [22] to increment the set of critical nodes. The authors conclude that no strategy dominates the others on the Solomon's instances. No computational study have been carried out on random generated networks. For these reasons, we consider the original algorithm presented in [120] for the comparison on Solomon's instances, whereas, for the instances of set S_2 we consider the original algorithn and the versions that make use of the strategies proposed by Boland et al. in [22]. In addition, we do not consider the strategy *All* for $\mathcal{DSSR}$ because it does not provide the entire set of non dominated solutions.

The data collected in the computational tests are reported in Tables 6.2-6.4. The best results obtained are highlighted in bold. For the sake of comprehension, we separately consider the computational results collected on the two sets of test problems.

6.8.2 Results on set S_1

In this Section we evaluate and compare the behaviour of $\mathcal{GSSAA}$ and $\mathcal{DSSR}$ in terms of computational effort (execution time) and memory occupancy (generated labels). The results are reported in Tables 6.2 and 6.3, where the number of additional node resources, the total number of generated labels and the time in ms are highlighted.

	$\mathcal{GSSAA}$			$\mathcal{DSSR}$		
test	$\lvert S\rvert$	labels	time	$\lvert S\rvert$	labels	time
c101_50_01	**0**	**1**	**0**	**0**	100	234
c101_50_02	**0**	**37**	**32**	**0**	100	219
c101_50_03	**0**	**241**	**1109**	**0**	1282	21578
c101_50_04	**1**	**2400**	28937	**1**	3089	**14875**
c101_50_05	**2**	**15319**	**314641**	**2**	21710	384265
c101_50_06	**3**	**72344**	**6296187**	5	80896	6903280
c101_50_07	**5**	**126535**	**6865219**	**5**	147962	8330640

continued on next page

continued from previous page

	$\mathcal{GSSAA}$			$\mathcal{DSSR}$		
test	$\|S\|$	labels	time	$\|S\|$	labels	time
c101_50_08	**7**	**139885**	**7200703**	**7**	152345	8546790
c101_50_09	**7**	**251249**	**10801625**	**7**	262435	13478905
c101_50_10	**7**	**258446**	**10802625**	**7**	304538	15630012
r101_50_01	**0**	**6**	**0**	**0**	126	390
r101_50_02	**0**	**14**	**15**	**0**	205	641
r101_50_03	**0**	**103**	**62**	**0**	587	5110
r101_50_04	**0**	**287**	**175**	**0**	800	1828
r101_50_05	**0**	**607**	**344**	**0**	1209	4015
r101_50_06	**0**	**1187**	**875**	**0**	1708	7797
r101_50_07	**3**	**26255**	**154078**	**3**	33398	876542
r101_50_08	**5**	**65758**	**2547297**	**5**	87904	3267008
r101_50_09	**5**	**136074**	**2547297**	**5**	187602	4236785
r101_50_10	**7**	**386701**	**8008094**	**7**	517893	10983450
rc101_50_01	**0**	**1**	**0**	**0**	100	234
rc101_50_02	**0**	**12**	**2**	**0**	100	204
rc101_50_03	**0**	**18**	**8**	**0**	231	735
rc101_50_04	**0**	**22**	**15**	**0**	231	750
rc101_50_05	**0**	**89**	**62**	**0**	541	1906
rc101_50_06	**2**	**1128**	**4016**	**2**	1915	3469
rc101_50_07	**5**	**12091**	**51547**	**5**	13305	135000
rc101_50_08	**5**	**83495**	**3364219**	**5**	100234	5623983
rc101_50_09	**6**	**104345**	**3899954**	**6**	123875	6239881
rc101_50_10	**7**	**131080**	**4594531**	**7**	158730	8872034
AVG	**2.57**	**60524.33**	**2249455.63**	2.63	73505.03	3119085.33

Table 6.2: Computational results collected on the instances with 50 nodes. $|S|$ indicates the number of additional node resources at the last iteration. In the column **labels** the total number of labels generated are reported, whereas the time in ms is shown in column **time**.

Since the behaviour of the algorithms seems to be different considering the instances with 50 and 100 nodes, first we discuss the computational results collected for the instances with 50 nodes.

Problems with 50 **nodes.** The collected data, reported in Table 6.2, underline that, on average $\mathcal{GSSAA}$ is 1.39 times faster than $\mathcal{DSSR}$. This behaviour is justified by the number of generated labels. In particular, $\mathcal{DSSR}$ generates a number of labels 1.21 times greater than those are generated by $\mathcal{GSSAA}$. However, when we consider the average execution time with respect to the vehicle capacity, some interesting trends can be underlined. The Figure 6.1 shows the average execution time of the two algorithms for each value of the vehicle capacity.

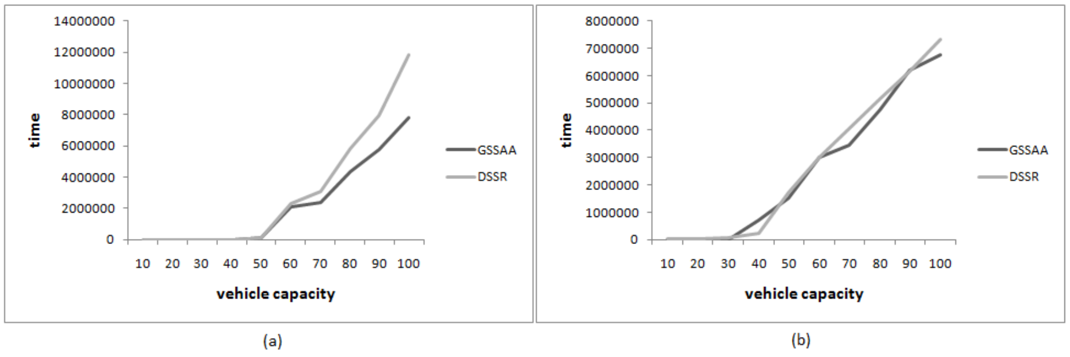

Figure 6.1: Average execution time as function of vehicle capacity for the instances with (a) 50 nodes and with (b) 100 nodes.

It is evident that for both $\mathcal{GSSAA}$ and $\mathcal{DSSR}$, the higher the vehicle capacity, the higher the computational effort. However, $\mathcal{GSSAA}$ requires less time than that required by $\mathcal{DSSR}$ (see Figure 6.1(a)). The only exeption can be observed for the instances with a vehicle capacity equal to 40. In addition, as shown in Figure 6.2(a), the gain in computational effort of $\mathcal{GSSAA}$ respect to $\mathcal{DSSR}$ increases when the capacity of the vehicle increases as well.

It is important to point out that the previous consideration does not allows us to claim that $\mathcal{GSSAA}$ is better than $\mathcal{DSSR}$ for a higher vehicle capacity. Indeed, if we consider the speed-up of $\mathcal{GSSAA}$ with respect to $\mathcal{DSSR}$, then a decreasing trend is observed when the capacity of the vehicle increases. This behaviour is depicted in Figure 6.3(a).

Regarding the memory occupancy, from Table 6.2 it is possible to observe that $\mathcal{DSSR}$ is slight worse than $\mathcal{GSSAA}$. Indeed, the average number of labels generated by the latter is 1.21 times less than that

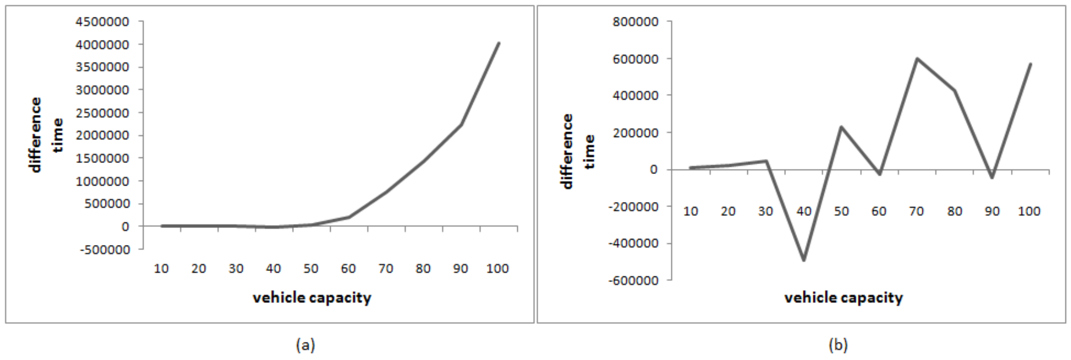

Figure 6.2: Difference between the computational time obtained with $\mathcal{DSSR}$ and $\mathcal{GSSA}$ for each value of the vehicle capacity, on instances with (a) 50 nodes and with (b) 100 nodes.

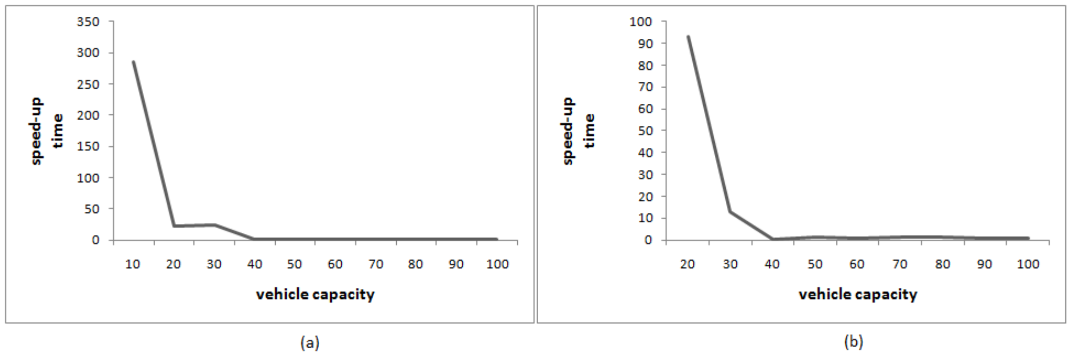

Figure 6.3: Speed-up of $\mathcal{GSSAA}$ respet $\mathcal{DSSR}$ for each value of the vehicle capacity on instances with (a) 50 and (b) 100 nodes.

generated by the former. As shown in Figure 6.4(a), it seems that the number of labels exponentially increases with the increasing of the vehicle capacity. This behaviour is observed for both $\mathcal{GSSAA}$ and $\mathcal{DSSR}$.

In addition, the higher the vehicle capacity, the higher the gap between the labels generated by $\mathcal{GSSAA}$ and those generated by $\mathcal{DSSR}$ (see Figure 6.5(a)).

Problems with 100 nodes. Table 6.3 shows the results collected on the instances with 100 nodes. On this set of instances $\mathcal{GSSAA}$ and $\mathcal{DSSR}$

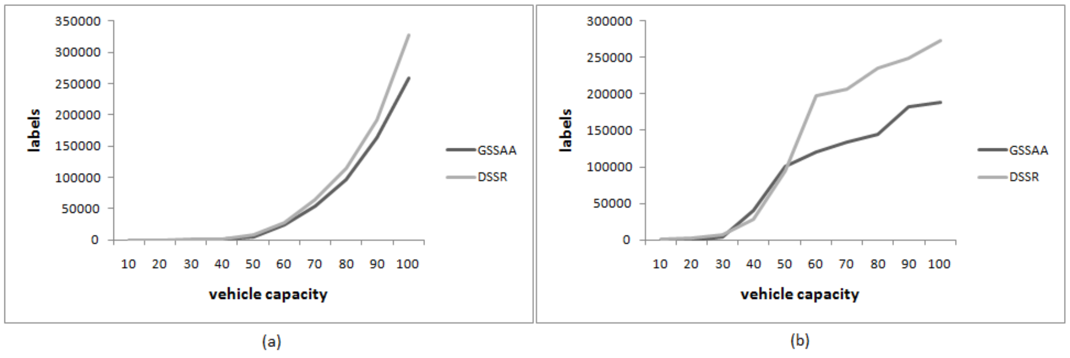

Figure 6.4: Average number of generated labels as function of vehicle capacity for the instances with (a) 50 nodes and with (b) 100 nodes.

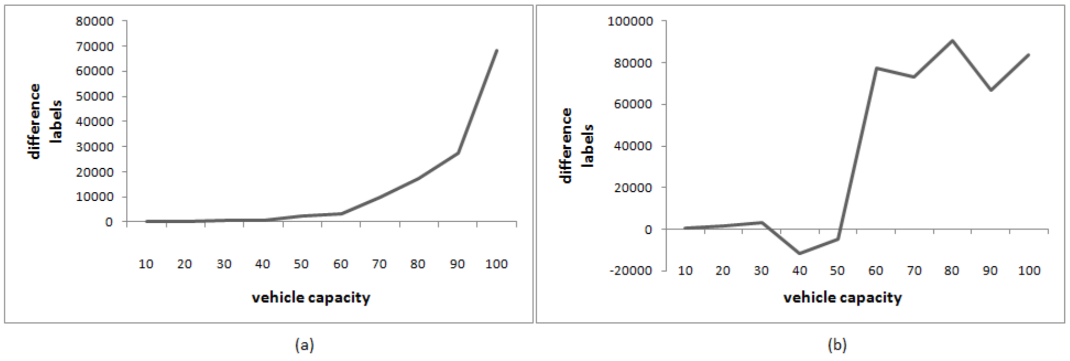

Figure 6.5: Difference between the labels generated by $\mathcal{DSSR}$ and $\mathcal{GSSAA}$ for each value of the vehicle capacity, on instances with (a) 50 nodes and (b) 100 nodes.

show similar performances. Indeed, the former is 1.05 times faster than the latter. From Figure 6.1(b), it is clear that the higher the vehicle capacity, the higher the time. In addition, it can be observed the similarity in the computational effort. Indeed, no algorithm dominates the other (see Figure 6.1(b)).

	$\mathcal{GSSAA}$			$\mathcal{DSSR}$		
test	$\lvert S\rvert$	labels	time	$\lvert S\rvert$	labels	time
c101_100_01	**0**	**1**	**0**	**0**	200	3078
c101_100_02	**0**	**11**	**16**	**0**	200	3172
c101_100_03	**0**	**64**	**109**	**0**	1136	6719

continued on next page

continued from previous page

	$\mathcal{GSSAA}$			$\mathcal{DSSR}$		
test	$\|S\|$	labels	time	$\|S\|$	labels	time
c101_100_04	**0**	**5372**	**14390**	**0**	1136	6906
c101_100_05	**0**	**2123**	**54344**	**0**	6388	537344
c101_100_06	**1**	**39491**	**3600141**	**1**	80451	3764280
c101_100_07	**1**	**43988**	**3601094**	**1**	84922	3892357
c101_100_08	**2**	**45677**	**4776547**	**2**	92505	5038127
c101_100_09	**2**	**50323**	**5202328**	**2**	97563	5237481
c101_100_10	**3**	**60862**	**5400718**	**3**	99873	6123845
r101_100_01	**0**	**20**	**31**	**0**	120	345
r101_100_02	**1**	**20**	**31**	2	2654	23547
r101_100_03	**2**	**8767**	**7766**	3	16407	96828
r101_100_04	**2**	109240	**2092281**	**2**	**80645**	557656
r101_100_05	**3**	289116	**4443203**	**3**	**274932**	4562819
r101_100_06	**3**	**310491**	5400141	**3**	436278	**5123822**
r101_100_07	**3**	**314413**	**5497875**	**3**	444562	6129736
r101_100_08	**4**	**327894**	**5967406**	**4**	517293	6325172
r101_100_09	**4**	**408019**	7200359	**4**	546372	**6825361**
r101_100_10	**4**	**412320**	**7201250**	**4**	612930	8236218
rc101_100_01	**0**	**15**	**16**	**0**	1270	30687
rc101_100_02	**2**	**456**	**625**	**2**	2437	35719
rc101_100_03	**2**	**1921**	**3235**	**2**	3033	41289
rc101_100_04	**2**	**4865**	**3516**	**2**	3455	76539
rc101_100_05	**2**	**9188**	**6219**	**2**	5123	93745
rc101_100_06	**3**	**13425**	**76251**	**3**	78233	101237
rc101_100_07	**4**	**43728**	**1247192**	**4**	92762	2121398
rc101_100_08	**4**	**62151**	**3472563**	**4**	97247	4134827
rc101_100_09	**5**	**87210**	**6156278**	**5**	101234	6371829
rc101_100_10	**6**	**92561**	7681728	**6**	104726	**7623962**
AVG	**2.17**	**91457.73**	**2636921.77**	2.23	129536.23	2770868.17

Table 6.3: Computational results collected on the instances with 100 nodes. $|S|$ indicates the number of additional node resources at the last iteration. In the column **labels** the total number of labels generated are reported, whereas the time in ms is shown in column **time**.

This behaviour can be justified by considering the difference in terms of time plotted in Figure 6.2(b). Indeed, on average, 3 out of 10 times, $\mathcal{DSSR}$ behaves better than $\mathcal{GSSAA}$. In addition, the higher the vehicle capacity, the higher the speed-up of the latter respect to the former.

When we consider the memory occupancy, $\mathcal{DSSR}$ generates a number of labels that is 1.42 times higher than that generated by $\mathcal{GSSAA}$. This behaviour is clearly shown in Figure 6.4(b). The gain in memory occupancy of $\mathcal{GSSAA}$ seems to increas when higher values of vehicle capacity are considered (see Figure 6.5(b)). It is important to observe that in 2 out of 10 cases, on average, $\mathcal{DSSR}$ generates a smaller number of labels than that generated by $\mathcal{GSSAA}$. Indeed, as shown in Figure 6.5(b), the gain in memory occupancy of $\mathcal{GSSAA}$ is negative for the instances with a vehicle capacity equal to 40 and 50.

6.8.3 Results on set S_2

The results collected on the second set of test problems are reported in Table 6.4, where the number of node resources at the last iteration, the memory occupancy in terms of total number of generated labels and the computational time are highlighted. Each line of the Table represents an average result, averaged on a group of ten problems, generated using identical parameters as input to the SPRAND generator.

It is important to point out that the two algorithms fail to solve some instances, thus in the following we refer only to the instances solved by both $\mathcal{GSSAA}$ and $\mathcal{DSSR}$. The instances that are not solved by the algorithms are highlighed in Table 6.4.

For the set S_2, we have applied the strategies proposed by Boland et al. in [22] to $\mathcal{DSSR}$. In particular, we have tested HMO, in which the first node with the highest multiplicity in the least cost path is added to the set of critical nodes, $HMOAll$, where all nodes with the highest multiplicity in the first feasible least cost path are marked as critical nodes and $MOAll$ that considers all the nodes with a multiplicity greater than one in the least cost path. It is important to point out that the strategies to increment the set $\mathcal{S}$ have never been considered for $\mathcal{DSSR}$ when random networks are tested.

From the data collected in Table 6.4, it is evident that the strategy HMO dominates both the $HMOAll$ and the $MOAll$. In particular, for $\mathcal{DSSR} - HMOAll$ and $\mathcal{DSSR} - MOAll$ the cardinality of the set $\mathcal{S}$ is 1.49 and 1.63 times higher than that for $\mathcal{DSSR} - HMO$. This bahaviour makes less efficient the strategies $HMOAll$ and $MOAll$ than the strategy HMO in terms of both the computational effort and the memory occupancy. Indeed, $\mathcal{DSSR} - HMO$ generates a numer of labels 1.21 and 1.29 times less than that generated by $\mathcal{DSSR} - HMOAll$ and $\mathcal{DSSR} - MOAll$, respectively. The efficiency of $\mathcal{DSSR} - HMO$ is more evident regarding the time. As a matter of the fact, $\mathcal{DSSR} - HMOAll$ and $\mathcal{DSSR} - MOAll$ are 2.12 and 2.46 times slower than $\mathcal{DSSR} - HMO$.

These results suggest that the strategy used in the original method of Righini and Salani ([120]) is not the most efficient when the random networks are considered. Indeed, $\mathcal{DSSR} - HMO$ shows the best performance in terms of both the memory occupancy and the computational time.

Regarding the comparison with $\mathcal{GSSAA}$, we consider the best performing version of $\mathcal{DSSR}$, that is that with HMO strategy.

From the data collected in Table 6.4, it is possible to observe that, on average, $\mathcal{GSSAA}$ behaves the best. Indeed, $\mathcal{DSSR} - HMO$ is 3.36 times slower than $\mathcal{GSSAA}$. This result is obtained even thougth the labels generated by the former is 1.31 times less than that generated by the latter.

This behaviour can be explained by considering that: 1) the bounded bi-directinal search strategy implemented in $\mathcal{DSSR}$ allows to terminate the extension of the labels obtaining a reduction of memory occupancy; 2) in addition, the reduction of generated labels does not suffice the computational effort to construct the paths starting from the forward and backward sub-paths.

It is worth observing that for the networks with a number of node less than 60, $\mathcal{DSSR} - HMO$ behaves the best. Indeed, considering the networks R_1 and R_2, the average number of labels generated by $\mathcal{GSSAA}$ is 1.93 times higher than that generated by $\mathcal{DSSR} - HMO$. Whereas, regarding the computational effort, $\mathcal{DSSR} - HMO$ is 1.14 times faster than $\mathcal{GSSAA}$. In Figure 6.6 the memory occupancy and the computational effort are plotted as a function of the percentage negative arc cost for both $\mathcal{DSSR} - HMO$ and $\mathcal{GSSAA}$.

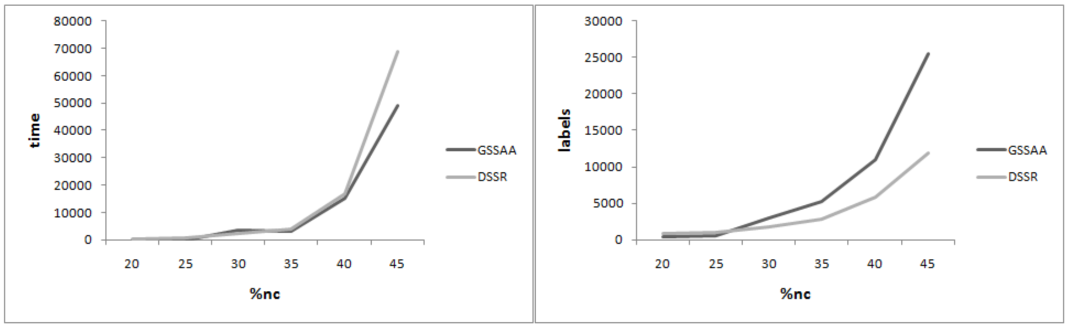

Figure 6.6: Average execution time and memory occupancy related to the instances with 30 and 40 nodes. The results are plotted based on the increasing order of the percentage negative arc cost.

In Figure 6.7, we report the number of labels and the time as a function of the density when the networks $R_3 - R_8$ are considered. Also in this case, $\mathcal{DSSR} - HMO$ behaves the best. Indeed, the average number of the labels generated by $\mathcal{DSSR} - HMO$ is 1.56 times slower than that generated by $\mathcal{GSSAA}$. This bahaviour makes $\mathcal{GSSAA}$ less efficient than $\mathcal{DSSR} - HMO$ in terms of computational effort. As a matter of the fact, $\mathcal{GSSAA}$ is 1.25 times slower than $\mathcal{DSSR} - HMO$.

Regarding the nerworks $R_9 - R_{12}$, $\mathcal{GSSAA}$ behaves the best. Indeed, $\mathcal{DSSR} - HMO$ is 13.99, 96.93 and 64.30 times slower than $\mathcal{GSSAA}$ for the networks with 60, 100 and 150 nodes, respectively. This behaviour can be justified by considering the numer of generated labels. Indeed, $\mathcal{DSSR} - HMO$ generates a number of labels that is 1.08, 1.46 and 2.27 times higher than that generated by $\mathcal{GSSAA}$ for the networks with 60, 100 and 150 nodes, respectively.

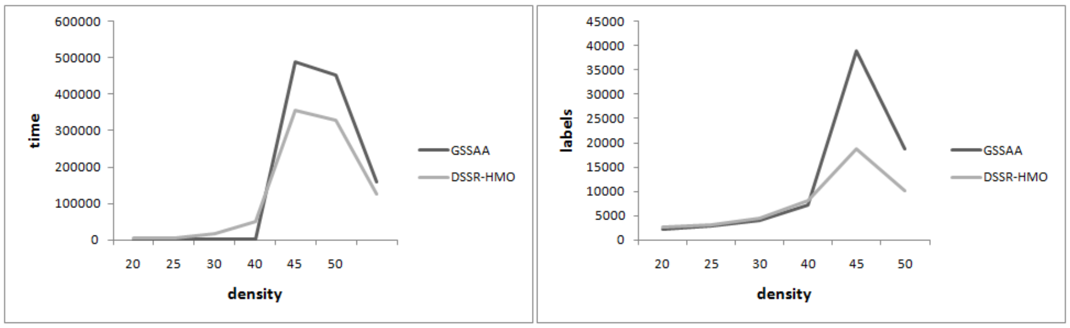

Figure 6.7: Average execution time and memory occupancy related to the instances with 50 nodes. The results are plotted based on the increasing order of the density.

	$\mathcal{GSSAA}$			$\mathcal{DSSR} - HMO$			$\mathcal{DSSR} - HMOAll$			$\mathcal{DSSR} - MOAll$		
test	$\|S\|$	labels	time	$\|S\|$	labels	time	$\|S\|$	labels	time	$\|S\|$	labels	time
R_1^{20}	0,20	**291,60**	**33,90**	0,20	457,20	228,70	**0,10**	457,20	224,80	**0,10**	457,20	228,20
R_1^{25}	**0,00**	**260,00**	**22,20**	**0,00**	454,70	195,30	**0,00**	454,70	208,70	**0,00**	454,70	195,20
R_1^{30}	**0,20**	1777,40	5784,80	**0,20**	**681,90**	**178,50**	0,70	894,20	524,30	0,70	894,20	550,80
R_1^{35}	**1,20**	6215,50	3521,40	**1,20**	**1516,90**	**878,20**	1,50	1765,00	1616,40	1,50	1765,00	1043,30
R_1^{40}	**2,30**	9315,60	7668,90	**2,30**	4266,80	**3280,90**	2,60	**3957,70**	4612,50	3,10	4344,20	7103,00
R_1^{45}	**2,00**	14833,40	34925,00	**2,00**	4919,70	**5053,20**	2,90	**4715,20**	10551,80	3,10	4968,50	12869,30
R_2^{20}	**0,30**	**607,80**	**31,30**	**0,30**	1082,00	529,30	0,40	1120,70	573,10	0,40	1120,70	575,90
R_2^{25}	**0,50**	**937,50**	**68,50**	**0,50**	1483,30	902,30	0,60	1537,00	1014,80	0,60	1537,00	1035,40
R_2^{30}	**1,20**	4215,10	**1360,70**	**1,20**	3135,00	3301,60	1,50	**2672,30**	3946,20	1,50	2672,30	4187,60
R_2^{35}	**1,40**	4172,60	**2523,40**	**1,40**	**3152,70**	3164,60	**1,40**	3895,80	6269,80	1,50	3763,70	6491,30
R_2^{40}	**1,70**	12577,50	22562,90	**1,70**	**6526,50**	**11409,50**	2,90	7618,40	28770,50	3,50	12385,50	163728,50
R_2^{45}	**3,50**	36156,00	**63222,00**	**3,50**	19657,50	94697,80	4,40	**19038,40**	126971,90	5,10	20732,90	185815,70
R_3^{15}	**0,10**	**541,90**	**35,60**	**0,10**	884,10	584,50	**0,10**	884,10	575,50	**0,10**	884,10	600,50
R_3^{20}	**0,00**	**511,50**	**30,10**	**0,00**	1009,20	703,80	**0,00**	1009,20	704,40	**0,00**	1009,20	731,80
R_3^{35}	**0,70**	**2710,50**	**267,00**	**0,70**	3283,50	4374,10	1,40	3609,70	7475,00	1,60	3781,70	9427,70
R_3^{40}	**1,10**	**4907,30**	**637,70**	**1,10**	5748,70	9843,30	2,00	6433,50	18737,40	2,10	6252,00	20157,90
R_4^{15}	0,20	**622,50**	**43,00**	0,20	1109,20	955,10	**0,20**	1109,20	952,80	**0,20**	1109,20	968,40
R_4^{20}	**0,30**	**952,00**	**79,70**	**0,30**	1646,10	1528,30	**0,30**	1646,10	1509,30	**0,30**	1646,10	1569,20
R_4^{30}	**0,90**	4586,80	**1816,60**	**0,90**	**4492,90**	8638,10	1,30	5395,60	34321,20	1,30	5127,70	35427,00
R_4^{35}	**1,00**	5401,40	**1299,90**	**1,00**	**4965,30**	10854,10	1,40	5915,80	20726,90	1,40	5915,80	21864,00
R_5^{15}	**0,10**	**630,30**	**54,60**	**0,10**	1324,50	1343,70	0,20	1392,90	1744,00	0,20	1392,90	1769,90
R_5^{20}	**0,30**	**996,20**	**81,50**	**0,30**	1897,40	2235,50	0,80	2352,60	3814,00	0,50	2053,60	3114,60
R_5^{30}	**1,00**	4708,80	**975,60**	**1,00**	**4636,80**	9112,20	1,20	5049,60	15402,90	1,20	5049,60	15991,70
R_5^{35}	**1,30**	**9860,50**	**4219,40**	**1,30**	10179,90	50512,10	1,90	13258,00	258539,90	2,40	18014,10	829145,20
R_6^{15}	**0,10**	**719,30**	**62,90**	**0,10**	1569,30	2184,50	0,30	1820,10	4125,60	0,30	1820,10	4198,80
R_6^{20}	**0,20**	**1170,00**	**123,70**	**0,20**	2119,30	3479,30	0,40	2395,50	6730,30	0,40	2395,50	7025,40
R_6^{30}	**1,60**	**9627,80**	**4170,40**	**1,60**	10449,10	58069,70	2,30	18525,00	626134,40	3,00	23430,10	1145063,80
R_6^{35}	**1,70**	**16875,10**	**8071,00**	**1,70**	18463,20	140004,20	3,20	27184,60	4840194,40	3,30	27626,90	1980628,30
R_7^{15}	**0,10**	**1076,10**	**206,00**	**0,10**	1729,40	2554,30	**0,10**	1729,40	2464,20	**0,10**	1729,40	2479,70
R_7^{20}	**0,40**	**1545,00**	**154,50**	**0,40**	2646,20	4880,50	0,60	2902,00	6399,30	0,60	2902,00	6532,70
R_7^{35}	$2,33^{(1)}$	52005,89	**131765,11**	$2,33^{(1)}$	**37028,50**	756213,50	3,20	37099,00	1400885,20	4,50	63496,90	5190308,90
R_7^{40}	**3,10**	101057,10	1825406,60	**3,10**	**33962,20**	**654231,60**	3,70	56592,60	3787754,80	$4,00^{(1)}$	43436,67	2062697,89
R_8^{15}	**0,10**	**882,80**	**90,80**	**0,10**	1809,40	2873,30	0,20	1932,50	3418,70	0,20	1932,50	3510,00
R_8^{20}	**0,30**	**1484,80**	**214,20**	**0,30**	2496,60	6428,40	0,30	2661,20	9033,30	0,40	2858,70	24054,10
R_8^{30}	**1,60**	**13677,40**	**17459,40**	**1,60**	14798,30	163701,40	2,70	24177,10	972578,00	2,90	25243,40	1051669,80
R_8^{35}	**1,90**	58807,30	1789888,00	**1,90**	**21462,20**	**1144418,40**	3,30	33920,70	2199813,60	3,30	32724,10	2302600,30
R_9^{15}	**0,00**	**711,40**	**78,70**	**0,00**	1435,60	2130,10	**0,00**	1435,60	2016,00	**0,00**	1435,60	2047,00
R_9^{20}	**0,00**	**814,50**	**78,00**	**0,00**	1659,80	2653,50	**0,00**	1659,80	2504,70	**0,00**	1659,80	2553,70
R_9^{35}	**1,60**	11865,30	**5668,90**	**1,60**	**11218,10**	59711,60	2,60	19013,80	592731,30	2,80	19490,00	654381,80
R_9^{40}	**1,20**	**12327,50**	**3889,30**	**1,20**	13511,10	71380,30	2,90	19198,90	511790,10	2,90	19198,90	562595,20
R_{10}^{15}	**0,00**	**1774,70**	**527,30**	**0,00**	3662,10	27850,90	**0,00**	3662,10	26736,70	**0,00**	3662,10	27619,40
R_{10}^{30}	**0,50**	**11204,30**	**7665,50**	**0,50**	14648,30	234059,80	1,20	22325,60	2089621,00	1,20	22325,60	2005496,90
R_{11}^{15}	**0,30**	**3971,10**	**2462,20**	**0,30**	7542,50	124602,00	0,40	7883,50	148169,60	0,40	7883,50	392332,60
R_{11}^{30}	**1,30**	**47408,60**	**92688,70**	$1,43^{(3)}$	67937,00	9630138,14	$2,33^{(4)}$	53652,33	10824854,67	$2,33^{(4)}$	53652,33	14379777,33
R_{12}^{15}	**0,00**	**3381,00**	**4626,50**	**0,00**	7664,40	297489,70	**0,00**	7664,40	294641,40	**0,00**	7664,40	302337,80
R_{12}^{30}	$1,22^{(1)}$	**144943,44**	**14789133,33**									
AVG	**0,89**	10670,59	**89923,63**	**0,89**	**8140,54**	302523,55	1,32	9858,19	642275,23	1,45	10531,12	742988,97

Table 6.4: Computational results collected on the instances of set S_2. $|S|$ indicates the number of additional node resources at the last iteration. In the column **labels** the total number of labels generated are reported, whereas the time in ms is shown

in column **time**. The superscript indicate the number of instances that are not solved by the algorithm. The entry OoM indicates that no problems are solved.

6.9 Conclusion and final remark

In this paper we have conducted a computational analysis of the two best known algorithms for the resource constrained elementary shortest path problem.

The algorithms considered in this work are the general state-space augmenting algorithm ($\mathcal{GSSAA}$, for short) proposed by Boland et al. in [22] and the algorithm of Righini and Salani ([120]), namely decremental state-space relaxation ($\mathcal{DSSR}$, for short). To the best of our knowledge, these algorithms turn out to be the most efficient methods to solve the problem.

This work has been motivated by several considerations that we summarize in what follows. 1) The problem at hand is one of the most important in network optimization. Indeed, the resource constrained elementary shortest path arises as sub-problem in column generation approach to solve more complicated problems, such as the vehicle routing problem with additional constraints and the short-haul aircraft rotation problem. 2) The scientific literature provides two resolution methods that are based on the same idea but that differ for some critical aspects. These issues have been described in detail in this work. 3) $\mathcal{GSSAA}$ and $\mathcal{DSSR}$ appeared in the scientific literature in the same period in two different papers. Thus, the computational exeperiments were conducted on different test problems.

In this paper, we have discussed about the similarity of the two algorithms and we have highlighted the differences related to critical aspects in terms of efficiency. For the computational studies, we have considered the instances proposed by the authors for testing their algorithm. Thus, we have unified the computational setting and the comparison between the $\mathcal{GSSAA}$ and the $\mathcal{DSSR}$ have been done on the same data-set. In this way, we have studied the behaviour of each algorithm not only on the test problems considered originally by the authors, but also on the instances considered by the others.

The test problems have been grouped in two sets containing the instances considered in [120] and those used by Boland et al. ([22]).

Our computational study suggests the following final results:

1) on the first set of test problems, that is that considered in [120], $\mathcal{GSSAA}$ and $\mathcal{DSSR}$ show similar performances. However, $\mathcal{GSSAA}$ behaves, on average, sligth better than $\mathcal{DSSR}$;

2) for the second set of test problems, on average, $\mathcal{GSSAA}$ behaves the best. In addition, some instances are not solved by both $\mathcal{GSSAA}$ and $\mathcal{DSSR}$. However, for the networks with a number of nodes less than 60, $\mathcal{DSSR}$ outperforms $\mathcal{GSSAA}$. The speed-up of $\mathcal{GSSAA}$ for the instances with a number of node from 60 to 150 is greater than the speed-up of $\mathcal{DSSR}$ for the networks with a number of nodes less than 60. This results justifies the best performance of $\mathcal{GSSAA}$ in the average case.

Chapter 7

Modelling and solving a multi-criteria path problem with multiple metrics and soft constraints [1]

Luigi Di Puglia Pugliese

Department of Electronics, Informatics and Systems, University of Calabria, 87036, Rende (CS), Italy

Janusz Granat

Institute of Control and Computation Engineering, University of Technology, Warsaw and National Institute of Telecommunications, Szachowa 1, Warsaw, Poland

Francesca Guerriero

Department of Electronics, Informatics and Systems, University of Calabria, 87036, Rende (CS), Italy

[1]Work presented at the EURO XXIII, the 23rd European Conference on Operational Research, Bonn, July 5-8, 2009

Abstract

A particular instance of the multi-criteria path problem is addressed in this work. In the considered problem it is assumed, that for each criterion, a lower and an upper bound are given. The problem is to find a path for which each criterion is the closest possible to the bounds. These constraints are taken into account by defining a specific objective function, using the concept of Chebychev distance. To solve the considered problem, a branch-and-bound method is developed. Extensive computational tests have been carried out on a meaningful number of random instances, with the purpose to assess the behaviour of the proposed approach in terms of robustness and efficiency.

Keywords: multiple criteria analysis; efficient paths; reference point method; branch and bound approach.

7.1 Introduction

The classical shortest path problem has been defined as a single-criterion problem, in which the main aim is to optimize a single objective, such as the total cost, the travel time ([20]).

However, in many real applications, arising in the design and management of different types of networks (i.e. transportation, transmission and communication networks), the complexity of the social and economic environment requires the explicit and simultaneous consideration of multiple and conflicting objectives (see e.g., [6], [14], [33], [41], [42]).

In this framework, unless a well-defined utility function exists, it is not possible to identify a single optimal solution, but rather different paths exist (i.e., Pareto-optimal paths) which can be considered as best solutions, in the sense that no improvement in any criterion is possible without sacrifice on at least one of the other criterion. Consequently, the decision maker should be able to get from the Pareto-optimal solution set a satisfactory compromise solution. While the single-criterion shortest path problem has a polynomial complexity bound ([20]), its multi-criteria counterpart is classified as NP-complete ([71]). Despite its theoretical complexity and given its practical importance, most studies have focused on the development of efficient solution approaches to address the multi-criteria shortest path problem. We cite [79], [103] and [128] where exact procedures such as multiple labelling methods have been proposed. Ranking methods to address the multi-criteria shortest path problem have been described in [10] and in [36], while parametric methods have been proposed in [110]. Approximate procedures have also been developed (see e.g., [85], [135], [142]). Other methods make use of utility (or cost) functions (e.g., [56], [108]). Interactive methods have been considered in [39], [40], [43], [74] and in [110].

In many real-life applications, objectives of different nature need to be optimized. For instances, when the time/cost is considered, the objective function is of additive type. When the objective represents a probability, then a multiplicative objective function has to be considered. In addition, in some real-life applications, criterion of bottleneck type occurs, that is bandwidth in the telecommunication field or machines capacity in the problems related to industrial production. It is worth observing that the multiplicative criteria can be transformed in additive type ([1], [139]). The multi-criteria shortest path with multiple metrics is a NP-complete problem only if at least two criteria are of additive type. Indeed, a special case, that is the bi-criteria shortest path problem where a criterion is of additive type and the other is a bottleneck criterion is proven to be easy to solve. The theoretical results on the complexity of the multi-criteria shortest path problem with multiple metrics are given in [137] and [139]. The scientific literature provides several works that address the multi-criteria shortest path problem when metric of additive and bottleneck types are taken into account simultaneously. In [116], the shortest path problem with two bottleneck and one additive criteria is addressed. Other solution approaches for the multi-criteria shortest path problem have been proposed in [11], [34], [140] and [141].

Nowadays people are more demanding. Consequently, the companies operating in services industry strive to provide high quality services to meet the increasing needs of the customers. In this context, it is important to provide high level of quality of services (QoS). This is true especially in the telecommunication industry.

It is important to observe, that the solution approaches proposed in the scientific literature to address the multi-criteria optimization problems arising in the telecommunication field, generally adopt one of the following strategy: 1) the different criteria are aggregate into a single objective function; 2) one metric to be optimized is chosen and upper bounds for the other metrics are imposed. For example, in the Interior Gateway Routing Protocol (IGRP) algorithm a combination (vector) of metrics is considered. In particular, the multi-attribute utility theory is used in [8] to address the routing problem in the telecommunication context. The authors make use of utility function that combines the criteria in such a way that the desires of the decision maker are achieved. Whereas, in the Open Shortest Path First (OSPF) routing algorithm, the cost is the single optimization criterion, but delay, throughput and connectivity are considered as additional constraints. Algorithms for solving routing problems that consider more than one criterion with constraints are defined in [21], [123], [147], [148].

The multi-criteria framework is used also to model several problems in the fields of transportation planning. In recent years there has been an increase in research on multi-criteria shortest path problem with multiple metrics, with goals of relevant interest, like the minimization of cost, time, risk and unreliability. For a review of multi-criteria shortest path problem the reader is referred to [42].

In this work we address the problem to find an efficient solution in a multi-criteria networks where constraints on each criterion are introduced. The goal is to find a solution for which each criterion is the closest possible to a specific value. The aspiration values represent the middle of a reference ranges defined by a lower and an upper bounds. In other words, it is assumed that the decision maker has specific idea about the solution that he/she wants to achieve.

The problem of finding the shortest paths for which each criterion belong to a specified range has been addressed in [34]. The proposed method allows to find the first k shortest paths and uses a reference point approach, where the paths in a specific priority region are ranked by non-decreasing order of a Chebyshev metric. In order to list paths according with this objective function a labelling algorithm is proposed. The solution approach provides a set of solutions that belong to a specified region (i.e., the region is defined by a lower and an upper bound for each criterion). If no solution exists in the specified region, the lower and upper bounds are modified to explore other decision spaces. A pruning procedure has been devised in order to reduce the number of generated labels. This procedure works only when the $k - st$ solution is found.

The aim of our work is to provide a solution approach that allows us to find exactly one solution that satisfies the reference values for each criterion. A branch-and-bound based approach is developed to optimality solve the problem under investigation.

The paper is organized as follows. In Section 7.2 we describe the problem and we give its mathematical formulation. The proposed solution approach is presented in Section 7.3. The effectiveness of the developed algorithm is tested on a large set of test problems and the related computational results are discussed in Section 7.4, while conclusions are given in Section 7.5.

7.2 Problem formulation

Let $G = (N, A)$ be a directed graph, where $N = \{1, \ldots, n\}$ is a finite set of nodes and $A \subseteq N \times N$ is a finite set of m arcs. Each arc is denoted by an ordered pair $(i,\ j)$, where $i \in N$ and $j \in N$. A path π_{ij} between the two distinct nodes $i \in N$ and $j \in N$ is defined as the sequence of nodes $\pi_{ij} = \{i = i_0, i_1, \ldots, i_{l-1}, i_l = j\}$ $l \geq 2$ such that $(i_h, i_{h+1}) \in A,\ h = 0, .., l-1$. The path is called *elementary* when no node is repeated in the sequence. When the nodes are repeated in the sequence we call it a *non-elementary* path.

With each arc $(i,\ j)\ \in A$, is associated a p-dimensional vector of criteria $q^{(i,j)} = \left(q_1^{(i,j)},\ q_2^{(i,j)}, \ldots, q_p^{(i,j)}\right)$, each of them representing a given metric.

A path π_{st} from the origin node $s \in N$ to the destination node $t \in N$ is evaluated by the p-dimensional vector of criteria $q^{\pi_{st}} = \left(q_1^{\pi_{st}},\ q_2^{\pi_{st}}, \ldots, q_p^{\pi_{st}}\right)$. Depending on the nature of the metric, the value $q_k^{\pi_{st}}$ of each criterion $k = 1, \ldots, p$ for the given path π_{st} from $s \in N$ to $t \in N$ can be defined in different way.

In what follows, we report a brief description of the most used metrics.

Additive metrics: in this case the value $q_k^{\pi_{st}}$ of the k-th criterion over the path π_{st} is the sum of $q_k^{(i,j)}$ on each arc in the path, that is $q_k^{\pi_{st}} = \sum_{(i,j)\in\pi_{st}} q_k^{(i,j)}$. Examples of additive metrics include cost, delay, jitter.

Multiplicative metrics: these are metrics for which the value $q_k^{\pi_{st}}$ of the criterion k on the path π_{st} is the product of $q_k^{(i,j)}$ on each arc belonging to π_{st}, that is $q_k^{\pi_{st}} = \prod_{(i,j)\in\pi_{st}} q_k^{(i,j)}$. Examples of multiplicative metrics include packet loss and error rate.

Bottleneck/concave metrics: the value of the k-th criterion $q_k^{\pi_{st}}$ over the path π_{st} is the value of the criterion on the bottleneck arc in the path, where the bottleneck arc is the arc in the path with the minimum value of the criterion, that is $q_k^{\pi_{st}} = \min_{(i,j)\in\pi_{st}} q_k^{(i,j)}$. Examples of bottleneck metrics include bandwidth.

It is worth observing that multiplicative metrics can be transformed in additive metrics ([1], [139]), for this reason in what follows we focus only on the additive and bottleneck type. It is important to point out that in this work we consider at least two metrics of additive type.

For each criterion $k = 1, \ldots, p$ an upper bound C_k^u and a lower bound C_k^l are given. It is worth observing that the upper and lower bounds are viewed as references given by the decision maker. He/She is satisfied by the solution whether the metrics achieve a value that is the middle of the range defined by the reference bounds. In other words, given a set of solution vector, the most satisfactory for the decision maker respect to his/her references is that belonging to the bound ranges and each component of such solution vector is the closest possible to the middle of the related range.

In this paper we formulate the aforementioned problem using a reference point concept, with the aim of optimizing the choice of the decision maker respect to his/her desires.

A reference point consists of desirable values for each objective function. In multiple objective mathematical programming, solution methods based on reference points can generate non dominated solutions using a variety of scalarizing functions ([106], [25]) that define the distance from the reference point. The idea is to take the desires of the decision maker into account when projecting the reference point onto the set of non dominated solutions. The procedures, to solve the problem formulated using the concept of reference point, are commonly referred to as reference point method.

We formulate the problem under consideration using the scalarizing function defined by Wierzbicki [143], see also [144]. In what follows, we briefly present some definitions related to the reference point method and we describe the property of the scalarizing function proposed in [143].

The reference point method is an interactive procedure that, given a set of controlling parameter $\overline{q}$, allows reaching an efficient solution. The selection of a particular efficient path is determined by the definition of the reference point (aspiration point). Most of those methods use the maximization of an Achievement Scalarizing Function (ASF). The ASF used in this work assumes the following form ([143]):

$$S\left(q^{\pi_{ij}}, \overline{q}, w\right) = \min_{1 \le k \le p} \left\{ w_k(\overline{q}_k - q_k^{\pi_{ij}}) \right\} + \epsilon \sum_{k=1}^{p} w_k(\overline{q}_k - q_k^{\pi_{ij}}) \tag{7.1}$$

where $q^{\pi_{ij}} \in R^p$ is the vector of criteria related to the path from node i to node j, $\overline{q} \in R^p$ is an aspiration point (reference point), $w_k > 0,\ k = 1, \ldots, p$, are scaling coefficients and ϵ is a given small positive number. Maximization is over all feasible paths from the origin node s to the destination node t. Maximization of (7.1) for all paths from s to t generates a properly efficient path with the trade-off coefficients smaller than $(1 + 1/\epsilon)$.

For a non-attainable $\overline{q}$, the resulting Pareto-optimal solution is the nearest - in the sense of a Chebyshev weighted norm - to the specified aspiration level $\overline{q}$. If $\overline{q}$ is attainable, then the Pareto-optimal solution is uniformly better. The selection of efficient paths is controlled by the vector $\overline{q}$. There is a common agreement that the aspiration point is a very good controlling parameter for examining a Pareto-optimal set.

The scaling coefficients w should not be confused with the weights used by some methods for conversion of a multi-criteria problem into a single-criterion problem with a weighted sum of original criteria. In the weighted sum methods, the weights are the main controlling parameters. In the reference point approaches, the scaling coefficients are needed to provide for uniform scaling of all criteria and are not used as controlling parameters.

The information provided by the user could be extended by specification of the reservation levels. These levels determine the values which the user would not like to achieve. In this cases the ASF usually takes the form:

$$(7.2)\quad S\left(q^{\pi_{ij}},\overline{q},\underline{q}\right)=\min_{1\le k\le p}\left\{f_k\left(q_k^{\pi_{ij}},\overline{q}_k,\underline{q}_k\right)\right\}\;+\epsilon\sum_{k=1}^{p}f_k\left(q_k^{\pi_{ij}},\overline{q}_k,\underline{q}_k\right).$$

where $\overline{q}$, $\underline{q}$ are vectors of aspiration and reservation levels respectively, and $f_k\left(q_k^{\pi_{ij}},\overline{q}_k,\underline{q}_k\right)$, $k=1,\ldots,p$ are the corresponding *Component Achievement Functions* ($CAFs$). Maximization of the function (7.2) over the set of feasible paths provides a properly Pareto-optimal solution with the properties discussed above for function (7.1).

In our case the aspiration and reservation levels represent the soft bounds. For each criterion $q_k,\ k=1,\ldots,p$ the decision maker prefers that its value belongs to the range $\left[\overline{q}_k=C_k^l,\ \underline{q}_k=C_k^u\right]$ and that is closest to the middle of the range. Thus, we propose the following $CAFs$:

(7.3)

$$f_k\left(q_k^{\pi_{ij}},\overline{q}_k,\underline{q}_k\right)=\begin{cases}aq_k^{\pi_{ij}}-a\overline{q}_k & if\ \ q_k^{\pi_{ij}}\le\ \overline{q}_k\\ q_k^{\pi_{ij}}-\overline{q}_k & if\ \ \overline{q}_k<q_k^{\pi_{ij}}\le\ \left((\overline{q}_k+\underline{q}_k)/2\right)\\ -q_k^{\pi_{ij}}+\underline{q}_k & if\ \ \left((\overline{q}_k+\underline{q}_k)/2\right)<q_k^{\pi_{ij}}\le\underline{q}_k\\ -aq_k^{\pi_{ij}}+a\underline{q}_k & if\ \ q_k^{\pi_{ij}}>\ \underline{q}_k\end{cases};k=1,\ldots,p;$$

where $a>1$. It is worth observing that functions (7.3) are not monotonic. We also underline that the solution should belong to a specific region defined by the aspiration and reservation levels. In addition, for each criterion, a solution with a value closest to the middle of the ranges is preferred. Functions (7.3) takes into account these considerations.

The multi-criteria shortest path problem with soft constraints can be formulated as follows:

$$\max_{\pi_{st}\in\Pi} S\left(q^{\pi_{st}}, \overline{q}, \underline{q}\right) \tag{7.4}$$

where Π is the set of all elementary paths. In the sequel we assume that $\epsilon = 0$. It is important to point out that solving problem (7.4), a non efficient solution could be found. In other words, the corresponding vector of objective functions is not Pareto-optimal. In order to explain this situation, let us consider the network of Figure 7.1. In the following example we suppose that all the criteria have to be minimized.

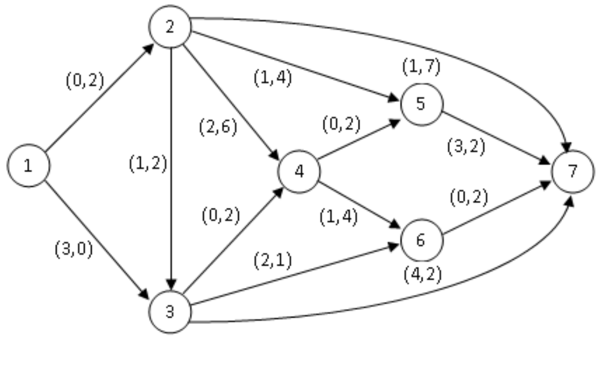

Figure 7.1: Graph example with two additive criteria. The vectors $(q_1^{(i,j)}, q_2^{(i,j)})$ represent the value of the first and the second criterion associated with the arc $(i, j) \in \mathcal{A}$, respectively.

The decision maker asks for a solution such that the lower and the upper bound for the first criterion is equal to 3 and 5, respectively; wherease for the second criterion, he/she wants a solution that belongs to the range $[8, 12]$. The ASF achieves the maximum value in the solution characterized by $q_1^{\pi_{17}} = 4$ and $q_2^{\pi_{17}} = 10$. In Figure 7.2, we report the criteria space and all the possible solutions from node 1 to node 7 in the graph example. It is easy to verify that the vector $(4, 10)$ is associated with a non efficient solution but it fullfills the requests of the decision maker.

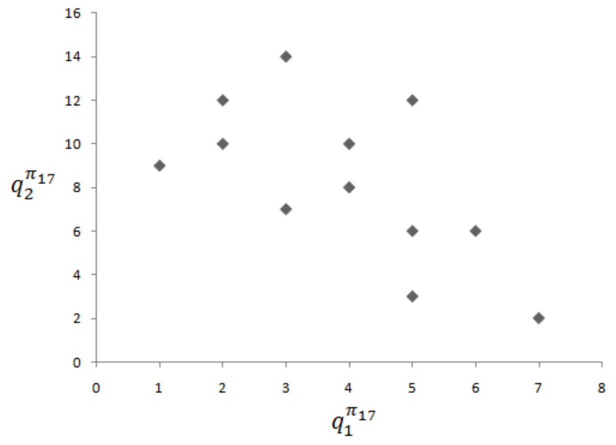

Figure 7.2: Graphical representation of the vector solutions associated with all possible solution from node 1 to node 7 in the network in Figure 7.1.

In the next section, we present the solution approach devised for solving problem (7.4).

7.3 Solution Approach

In this section we describe the solution method defined to solve the problem under investigation. It is worth observing that, the label correcting approach proposed in [74] cannot be applied to solve (7.4), since this method is based on the monotonic form of the $CAFs$ and this property is not satisfied in our case. Indeed, a label correcting algorithm cannot be used to solve the problem under consideration, since, given the non monotonicity of the $CAFs$, an optimal partial path is not necessary part of the optimal solution. Let us consider the network of Figure 7.1.

We assume that $\overline{q}_1 = 2$, $\underline{q}_1 = 5$, $\overline{q}_2 = 2$ and $\underline{q}_2 = 4$. We want to solve the problem from the origin node $s = 1$ to the destination node $t = 7$. When the label correcting algorithm is applied, the path π_{17} is determined (see Figure 7.3).

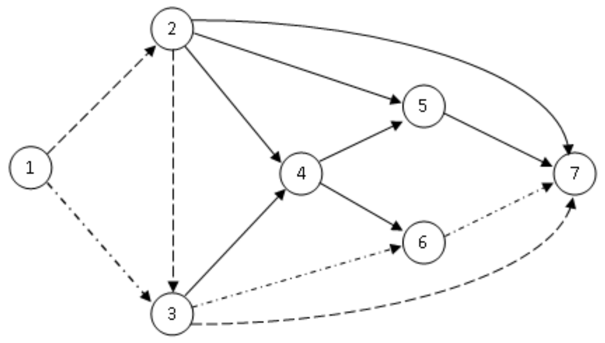

Figure 7.3: In dashed line is reported the path π_{17} obtained with the label correcting algorithm. In dotted line the path $\widetilde{\pi}_{17}$ is depicted.

The path π_{17} is characterized by the solution vector $q^{\pi_{17}} = [5,\ 6]$. The ASF evaluated for π_{17} is equal to -3. On the network of Figure 7.1, another solution exists, that is $\widetilde{\pi}_{17}$ with a corresponding vector $[5,\ 3]$. The built path is reported in Figure 7.3.

The value of the ASF for $\widetilde{\pi}_{17}$ is equal to 0 that is better than that achieved for π_{17}. It is evident that the path $\widetilde{\pi}_{17}$ represents the optimal solution, thus the solution found by the label correcting algorithm is sub-optimal. In order to understand better the reason why the label correcting algorithm finds a sub-optimal solution, let us consider node 3. The value of the ASF for the path π_{13} is equals to -1.5, while for the path $\widetilde{\pi}_{13}$ the ASF is equals to -3. In the course of the algorithm, node 2 is chosen as predecessor of node 3 because the ASF is better than the choice of node 1 as predecessor of node 3. In other words, an optimal partial solution could not be part of the optimal solution. In the sequel, we refer to the partial optimal solution that could not be part of the optimal solution as *local optimum*.

7.3.1 Branch and Bound approach

In order to optimality solve the problem under consideration, a branch-and-bound (B&B, for short) based algorithm is developed. The proposed approach relies on the determination of a lower bound on the optimal solution. The lower bounds are obtained by applying the label correcting algorithm ([74]) for solving problem (7.4). In order to evaluate the optimality of the solution, a set of problems are determined and solved.

Branching rule

In order to describe the branching rule, let P_i be a generic problem of the B&B tree defined on the directed graph $G^i = (N, A^i)$ and let $\pi^{P_i} = \{s = i_0, i_1, \ldots, i_{l-1}, i_l = t\}$, $l \geq 2$ such that $(i_j, i_{j+1}) \in A$, $j = 0, .., l-1$ be the path obtained by solving P_i. The corresponding vector of criteria $q^{\pi^{P_i}} = \left(q_1^{\pi^{P_i}},\ q_2^{\pi^{P_i}}, \ldots, q_p^{\pi^{P_i}}\right)$ is associated with path π^{P_i}.

From problem P_i, $l-1$ sub-problems are generated. Each sub-problem P_{i+n}, $n = 1, \ldots, l-1$ is defined on the graph $G^{i+n} = \left(N, A^{i+n}\right)$, where $A^{i+n} = A^i - \{(i_{n-1}, i_n)\}$.

The branching rule allows us to check if *local optimum* diverts the search process of the label correcting algorithm toward a non optimal solution. In addition, the proposed strategy ensures the exploiting of the entire solution space. These results are formally stated in the following Theorem.

Theorem 7.3.1. *The branching procedure leads to an exhaustive search of the solutions space.*

Proof. The branching procedure generates a set of problems defined on graph G^{i+n}, obtained by G^i where the arc (i_{n-1}, i_n) is removed. This means that from node i_{n-1} is not possible to reach directly node i_n. In the father graph G^i the label correcting algorithm has determined the best path from node s to node t. When the arc (i_{n-1}, i_n) is removed from the graph G^i, obtaining the graph G^{i+n}, the search process of the label correcting algorithm is constrained to reach node i_n without passing across node i_{n-1}. A partial path $\pi_{si_n}^{P_{i+n}}$ is obtained on the graph G^{i+n} from node s to node i_n. The following two different cases can occur:

Case 1. $f(\pi_{si_n}^{P_{i+n}}) \leq f(\pi_{si_n}^{P_i})$ or

Case 2. $f\left(\pi_{si_n}^{P_{i+n}}\right) > f(\pi_{si_n}^{P_i})$.

Case 1. Let us consider the first case.

Let $\pi^{P_{i+n}}$ be the path generated on G^{i+n} and suppose that $i_n \in \pi^{P_{i+n}}$.

If $f\left(\pi^{P_i+n}\right) > f(\pi^{P_i})$ then a *local optimum* is found and removed, otherwise (i.e., $f(\pi^{P_i+n}) \leq f(\pi^{P_i})$) $f(\pi_{si_n}^{P_i})$ is the optimum (non local) partial solution ending to node i_n.

Let us suppose that $i_n \notin \pi^{P_i+n}$. We can have two situations:

$$f(\pi^{P_{i+n}}) \geq f(\pi^{P_i}); \tag{7.5}$$

$$f\left(\pi^{P_{i+n}}\right) < f\left(\pi^{P_i}\right). \tag{7.6}$$

First, we consider condition (7.5). A path $\pi^{P_{i+n}}$ that does not contain node i_n exists. Thus, this path was also in G^i (we removed only the arc (i_{n-1}, i_n)). In other words, the optimal path in the graph G^i is $\pi^{P_{i+n}}$. For this reason we can say that the (7.6) is not valid.

If condition (7.6) is verified, $f(\pi_{si_n}^{P_i})$ is the optimum (non local) partial solution to node i_n.

Case 2. $f\left(\pi_{si_n}^{P_{i+n}}\right) > f(\pi_{si_n}^{P_i})$.

If $f\left(\pi^{P_i+n}\right) > f(\pi^{P_i})$ then we found a better solution that is candidate to contain the optimal solution.

If $f(\pi^{P_i+n}) \leq f(\pi^{P_i})$ then the label correcting algorithm solving P_i does not fall in local optimum. In G^{i+n}, removing arc (i_{n-1}, i_n), we impose to fall in the local optimum. In this case the problem P_{i+n} is fathomed.

With the branching rule, for all node $j \in \pi^{P_i} - \{s\}$ if the local optimum is found, then a better solution is determined. This new solution is candidate to be the optimal one. If a local optimum on $j \in \pi^{P_i} - \{s\}$ is not found and if the considered solution improves the objective function determined so far, then it is candidate to be the optimal solution. If the existing solution is better than the considered one, the related problem is aborted.

In this way we have an exhaustive search of the solution space. □

We said that the branching procedure allows to check if *local optimum* exists on each node $j \in \pi^{P_i} - \{s\}$. From Theorem 7.3.1, we know that if for problem P_{i+n} the partial path to reach node i_n is not *local optimum*, then the branching rule is not applied. On the other hand, a better solution is found, thus the local optimum for node i_n is removed. In addition, it is necessary to evaluate the optimality of the better solution, thus the branching rule is applied

In formal way, if $f\left(\pi^{P_{i+n}}\right) > f(\pi^{P_i})$ then the branching rule is applied for P_{i+n}, otherwise, the problem P_{i+n} can be fathomed.

Algorithm and its properties

In this section, we formalize the B&B based solution approach devised for solving the problem under investigation. Let $z^{(m)}$ be the incumbent at iteration m, that is the best solution determined so far and let $L^{(m)}$ be the list of candidate problems to be solved at iteration m.

The steps of the proposed method are depicted in **Algorithm** 20.

Algorithm 20 B&B approach

```
Step 0 (Initialization)
$L^{(0)} = \{P_0\}$, $P_0$ is defined on $G^0 = (N,\ A^0)$, where $A^0 \equiv A$; $z^{(0)} = -\infty; k = 0$.

Step 1 (Problem selection)
Select from $L^{(k)}$ a problem $P_i$ and delete it from $L^{(k)}$.

Step 2 (Branching procedure)
Branch the problem $P_i$ in $n = 1, \dots, l-1$ sub-problem $P_{i+n}$ defined on $G^{i+n} = (N,\ A^{i+n})$, where $A^{i+n} = A^i - \{(i_{n-1}, i_n)\}$, $(i_{n-1}, i_n) \in \pi^{P_i}$
Solve the sub-problem.

Step 3 (Updating)
Set $z^{(k+1)} = z^{(k)}$.
for all $P_{i+n}$ do
  if $f\left(\pi^{P_{i+n}}\right) > f(\pi^{P_i})$ then
    $L^{(k+1)} = L^{(k)} \cup P_{i+n}$.
    if $f\left(\pi^{P_{i+n}}\right) > z^{(k+1)}$ then
      $z^{(k+1)} = f\left(\pi^{P_{i+n}}\right)$.
    end if
  end if
end for
$k++$.
Go to Step 1.

Step 4 (Termination check)
if $L^{(k)} = \emptyset$ then
  STOP $z^{(k)}$ is the optimal solution.
else
  Go to Step 1.
end if
```

The devised solution approach allows us to find the optimal solution for the multi-criteria shortest path problem with soft constraints. In what follows, we formalize the optimality condition.

Theorem 7.3.2. *If $L^{(M)}$ is empty, then $z^{(M)}$ is the optimal solution.*

Proof. Let suppose to be at iteration m. A problem P_i is selected from $L^{(m)}$. All the sub-problem P_{i+n} are genereted and solved. Applying the bounding rule, if $f\left(\pi^{P_{i+n}}\right) \leq f\left(\pi^{P_i}\right)$, then P_{i+n} is fathomed. Wherease, if $f\left(\pi^{P_{i+n}}\right) > f(\pi^{P_i})$, then the set of candidate problems is updated, that is $L^{(m)} = L^{(m-1)} \cup P_{i+n})$. All the problems belonging to $L^{(m)}$ have the potential to contain the optimal solution, thus $z^{(m+1)} = \max_{P \in L^{(m)}} \left\{f\left(\pi^P\right)\right\}$ and $z^{(m+1)} > z^{(m)}$. This happens because $z^{(m)} = \max_{P \in L^{(m-1)}} \left\{f\left(\pi^P\right)\right\}$. Let suppose that $\tilde{P}_i = arg\max_{P \in L^{(m-1)}} \left\{f\left(\pi^P\right)\right\}$, consequently $z^{(m)} = f\left(\pi^{\tilde{P}_i}\right)$.

Let us consider the following two different situations.

1. $\tilde{P}_i$ is chosen at iteration m.

Let us suppose that $\tilde{P}_{i+n'} = arg\max\left\{f\left(\pi^{P_{i+n}}\right), f\left(\pi^{P_{i+n+1}}\right), \ldots.\right\}$. If $f\left(\pi^{\tilde{P}_{i+n'}}\right) \geq f\left(\pi^{\tilde{P}_i}\right)$, then $f\left(\pi^{\tilde{P}_{i+n'}}\right) \geq z^{(m)}$. A better solution is found, thus $z^{(m+1)} = f\left(\pi^{\tilde{P}_{i+n'}}\right)$ and all the problems such that $f\left(\pi^{\tilde{P}_{i+n}}\right) > f\left(\pi^{\tilde{P}_i}\right)$ are added to the list of candidate problems. On the other hand, if $f\left(\pi^{\tilde{P}_{i+n'}}\right) \leq f\left(\pi^{\tilde{P}_i}\right)$, then all the problems $\tilde{P}_{i+n}$ are fathomed and $z^{(m+1)} = z^{(m)}$.

2. $P_i \neq arg\max_{P \in L^{(m-1)}} \left\{f\left(\pi^P\right)\right\}$ is chosen at iteration m.

The sub-problems P_{i+n} are generated and solved. Also in this case if $f\left(\pi^{P_{i+n}}\right) \leq f\left(\pi^{P_i}\right)$ $n = 1, \ldots, l-1$, then the sub-problem is fathomed, otherwise if $f\left(\pi^{P_{i+n}}\right) > f\left(\pi^{P_i}\right) n = 1, \ldots, l-1$, then P_{i+n} is added to the list of candidate problems and if $f\left(\pi^{P_{i+n}}\right) > z^{(m)}$ for some n, then $z^{(m+1)} = f\left(\pi^{P_{i+n}}\right)$.

In the two cases the incumbent is updated by the value of the objective function of the problem that are stored in L. The problems that are fathomed have a less value of the objective function than that of the father problem. We know that $z = \max_{P \in L} \left\{ f\left(\pi^P\right) \right\}$, thus $z \geq \max_{P \in W} \left\{ f\left(\pi^P\right) \right\}$ where W is the set of all fathomed problems. It is evident that, when $L = \emptyset$, z is the best value among all the problems that belong to $L^{(m)}$, $m = 0, \ldots, M$ where M is the last iteration and $z^{(M)} \geq \max_{P \in W^{(M)}} \left\{ f\left(\pi^P\right) \right\}$, thus $z^{(M)} \equiv z^*$. □

The strategy for selecting the next sub-problem to investigate determines how the B&B algorithm should proceed through the search tree. The order, in which the nodes are processed, determined by the search strategy, can have a signicante effect on the behaviour of the algorithm.

In the proposed approach the selected sub-problem is that with the best bound; e.g., the highest lower bound. This selection strategy is motivated by the observation that the sub-problem with the best lower bound has to be evaluated anyway and that is more likely to contain the optimal solution than any other node (sub-problem in branch tree). As shown in [87], this strategy has the characteristic that, if other parts of B&B algorithm are not changed, the number of sub-problems decomposed before termination is minimized.

7.4 Computational experiments

In this section we evaluate the behaviour of the proposed solution approach in terms of robustness and efficiency. The B&B algorithm has been coded in Java language and the tests have been carried out by using an intel(R) core(TM) i7 cpu M620, 2.67 GHz, ram 4.00 GB, under Microsoft 7 operating system.

7.4.1 Test problems

The computational experiments have been conducted by considering several scenarious. In particular, different types of networks have been considered and we have simulated various preferences of the decisor maker. Regarding the choise of the networks, we have considered four sets of test problems. Random networks of varying size and density (defined as the ratio between the number of arcs and the number of nodes) have been considered. In particular, the developed approach has been tested on fully random networks, generated by using the public domain SPRAND generator ([29]). In set $S1$, we include the random generated networks with a small size ($R_1 - R_{10}$), whereas networks $R_{11} - R_{26}$, that have higher dimensions are included in the set $S2$. The characteristics of the considered random generated networks are reported in Tables 7.1 and 7.2.

test	nodes	arcs	density
R_1	300	2000	6.67
R_2	350	2000	5.71
R_3	400	2000	5.00
R_4	450	2000	4.44
R_5	500	2000	4.00
R_6	300	3000	10.00
R_7	350	3000	8.57
R_8	400	3000	7.50
R_9	450	3000	6.67
R_{10}	500	3000	6.00

Table 7.1: Characteristics of the random generated networks of the set $S1$
.

The set $S3$ contains the Solomon's complete networks. The original data set proposed by Solomon is divided into random, clustered, and random-clustered categories, according to the displacement of the customers. Instances belonging to the same data-set have the customers located in the same way and with the same delivery requests; these instances differ only for the time windows. Since we are inter-

test	nodes	arcs	density
R_{11}	100	5000	50.00
R_{12}	200	5000	25.00
R_{13}	500	5000	10.00
R_{14}	1000	5000	5.00
R_{15}	200	10000	50.00
R_{16}	400	10000	25.00
R_{17}	1000	10000	10.00
R_{18}	2000	10000	5.00
R_{19}	400	20000	50.00
R_{20}	800	20000	25.00
R_{21}	2000	20000	10.00
R_{22}	4000	20000	5.00
R_{23}	600	30000	50.00
R_{24}	1200	30000	25.00
R_{25}	3000	30000	10.00
R_{26}	6000	30000	5.00

Table 7.2: Characteristics of the random generated networks of the set $S2$

.

ested only in the structure of the networks, thus one instance taken from each one of the three Solomon's data-sets is considered. In particular, we have considered the original networks with 100 nodes and others by considering the first 50 nodes. We refer to these test problems as c_{101}, r_{101} and rc_{101} for the networks with 100 nodes, whereas we indicated with c_{51}, r_{51} and rc_{51} those with 50 nodes.

In addition, we have considered the set $S4$ containing 7 complete networks (i.e., $C_1 - C_7$) built by using the Compligen generator of Bertsekas ([20]). In Table 7.3 we report the number of nodes and the number of arcs of the considered complete networks that belong to the set $S4$.

For the networks of sets $S1$, $S2$ and $S4$ the costs have been randomly generated from the interval $[0, 100]$. For each network, we have generated a set of instances varying both the middle of the range that defines the preferences of the decisor maker and the width of the range. In particular, $\overline{q}_k$ and $\underline{q}_k$, $\forall k = 1, \dots, q$ are computed as $q_k^{\pi^*_{st}} + const - \gamma$

test	nodes	arcs
C_1	50	2450
C_2	80	6320
C_3	100	9900
C_4	150	22350
C_5	180	32220
C_6	200	39800
C_7	250	62250

Table 7.3: Characteristics of the complete networks of the set $S4$

.

and $q_k^{\pi^*_{st}} + const + \gamma$, respectively. The value $q_k^{\pi^*_{st}}$, $\forall k$, represents the cost of the optimal path when only the criteria k is optimized; the parameter *const* is a given constant; whereas the value γ is randomly chosen according to an uniform distribution in a specific range. In this work we consider 5 values of *const*, that is 150, 225, 300, 350, and 450. In addition, we have taken into account 3 different ranges in which γ is chosen. In particular, the ranges $r1 = [100; 1000]$, $r2 = [100; 3000]$ and $r3 = [100; 5000]$ have been considered. In what follows, we refer to the instances generated from networks R_i with γ chosen in the j-st range as R_i^{rj}, $j = 1, \ldots, 3$.

7.4.2 Experimental results

The data collected in the computational phase are reported in Tables A.1 - A.5 of the Appendix, where the computational time in ms, the number of iterations executed by the B&B approach and the B&B nodes generated are highlighted. Each line of a table represents an average result, averaged over a group of four different instaces that differ for the value of γ.

The computational results underline that the behaviour of the proposed solution approach is related to the width of the range defined by the decisor maker for each criterion. Indeed, the higher the interval, the higher the computational cost. This behaviour is observed for the instances of all the considered sets. In Figure 7.4, the trends of the computational time, of the number of iterations and of the number of B&B nodes are plotted as a function of the range.

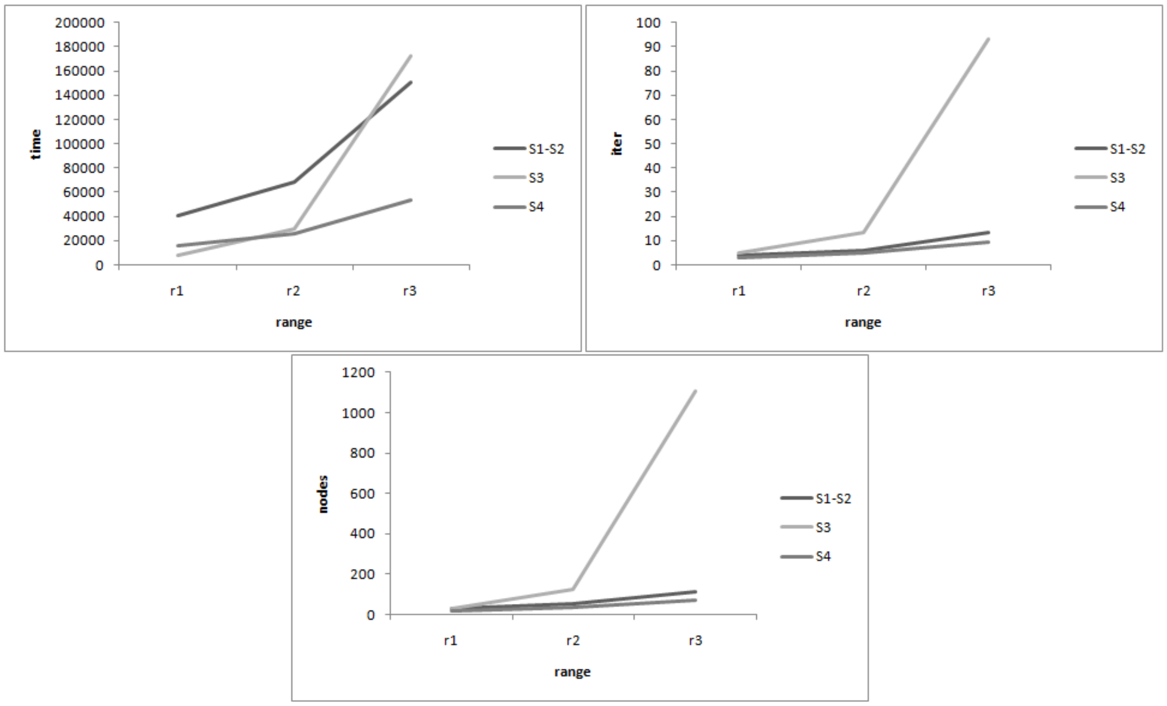

Figure 7.4: Time, number of iterations and number of nodes plotted as a function of the range.

As shown in Figure 7.4, the easiest problems to be solved are those belonging to the set $S4$. Indeed, the execution time of proposed solution approach in solving the instances of sets $S3$ and sets $S1 \cup S2$ is 2.22 and 2.73 times slower than the times required to solve the networks of set $S4$. As a matter of the fact, the number of iterations [nodes] executed [generated] is 1.36 and 6.56 [1.62 and 10.23] times higher than those obtained on set $S4$ for the sets $S1 \cup S2$ and $S3$, respectively. It is important to point out that the computational effort per node is higher for the set $S1 \cup S2$ than for the others sets. In particular, the time per node is equal to 1287.52, 216.94 and 774.10 for the set $S1 \cup S2$, $S3$ and $S4$, respectively.

No relationship can be observed beetwen the behaviour of the proposed algorithm and the value of the parameter *const*.

range	density	time	iter	nodes	time par node
r1	50	17022.55	2.79	20.03	850.06
	25	18712.56	2.29	15.73	1189.99
	10	58198.23	2.40	21.01	2769.70
	5	137119.49	2.36	23.16	5919.89
r2	50	19431.00	2.88	21.23	915.48
	25	30989.51	3.66	28.65	1081.66
	10	103865.66	4.85	45.48	2284.02
	5	250836.03	9.33	89.93	2789.39
r3	50	14989.41	4.68	34.30	437.01
	25	56446.56	10.88	84.26	669.89
	10	155421.43	10.51	90.29	1721.41
	5	685411.69	24.87	224.85	3048.30

Table 7.4: Computational effort, number of iterations, number of generated nodes and time par node when the networks of set $S2$ are considered, grouped by the value of the density.

Regarding the sets $S1$ and $S2$, we can drawn some considerations related to the structure of the random generated networks. In particular, it seems that the performance of the proposed strategy to solve the problem at hand is influenced by the dimensions of the test problems and by the density. Indeed, from Table 7.4 it is evident that for the networks of the set $S2$, the lower the density, the higher the computational effort. This behaviour can be justified by considering the number of iterations and the number of nodes generated when the instances of set $S2$ are solved (see Table 7.4). In addition, the computational effort per node increases when the density descreases.

When we consider the networks of the set $S1$, this trend is not observed. In others words, for the instances with a small size, that is those belonging to the set $S1$, there are no relations beetwen the behaviour of the proposed solution approach and the structure of the network.

7.5 Conclusions

In this work we have investigated a path problem in presence of multiple metrics and preferences of the decisor maker. In particular, for each metric, the decisor maker provides a range of values in which the function related to the metric should belong. We have modeled these preferences as soft constraints. In other words, if a solution for which each criterion belongs to the given range does not exists, then the solution the closest to the preferences of the decisor maker is found. To mathematically represent the problem, we have used the concept of reference point. An appropriate scalarizing function has been defined in order to take into account the desiders of the decisor maker. In particular, this function gives the distance of the solution from the reference point that is defined on the bases of the preferences of the decisor maker. Thus, the obtained solution is as close as possible to the desiders of the decisor maker. It is worth mentioning that the optimal solution is not necessary Pareto optimal. A non efficient solution could be optimal for the problem under investigation because of the particular demands of the decisor maker.

A branch-and-bound based solution approach is devised in order to optimally solve the problem under investigation.

The computational phase has been carried out by considering different scenarious. Indeed, several types of networks and different possible desiders of the decisor maker have been considered. The experimental results suggest that the performance of the proposed strategy is closely related to the width of the range, the higher the width, the higher the computational effort. No relationship has been observed beetwen the behaviour of the proposed method and the value of the middle of the ranges.

Appendix A - Computational results

const	150			225			300			350			400		
test	time	iter	nodes	time	iter	nodes	time	iter	nodes	time	iter	nodes	time	iter	nodes
R_1^{r1}	2067.00	3.00	19.25	6013.75	6.25	45.25	4098.75	3.00	25.75	71175.50	4.75	49.25	3617.75	1.00	19.00
R_2^{r1}	3956.50	6.00	35.50	1993.25	1.75	13.75	2903.00	2.00	17.50	2537.00	1.25	13.00	13948.25	7.25	81.50
R_3^{r1}	22202.75	30.50	204.50	1614.75	1.00	10.00	1692.75	1.00	10.25	1786.50	1.00	10.75	2784.50	1.50	17.00
R_4^{r1}	1229.50	1.25	9.25	15032.75	11.50	90.75	3237.25	1.75	17.50	3695.50	1.50	17.00	4050.75	1.00	15.50
R_5^{r1}	2745.75	2.50	19.00	2402.25	1.50	13.25	9629.00	4.75	49.50	28263.25	10.50	116.25	9656.50	4.00	40.25
R_6^{r1}	2574.00	2.00	15.25	2609.00	1.50	13.00	2194.25	1.00	9.00	4609.75	2.00	19.75	7256.50	3.50	32.75
R_7^{r1}	2510.50	1.75	13.00	2732.00	1.25	11.75	4187.25	1.75	17.25	5780.75	2.25	23.00	3613.00	1.00	13.25
R_8^{r1}	5078.50	5.00	27.25	4315.25	2.75	21.00	33692.00	18.75	145.75	5809.00	2.25	19.50	6823.25	1.75	21.25
R_9^{r1}	2185.00	1.25	9.25	7021.25	3.50	28.25	8052.25	3.50	31.75	26909.00	7.25	83.50	13354.50	3.50	39.50
R_{10}^{r1}	5070.50	2.50	20.25	7062.75	3.50	26.75	38495.50	16.25	133.75	205226.25	82.25	741.25	3727.25	1.00	10.25
R_{11}^{r1}	1591.50	1.75	8.75	2457.00	1.50	10.25	2898.75	1.75	12.00	5460.00	3.75	26.75	3378.25	1.25	10.25
R_{12}^{r1}	1876.00	1.50	8.50	1890.00	1.25	7.25	2691.50	1.25	8.75	2936.75	1.25	9.75	3253.50	1.00	8.50
R_{13}^{r1}	2143.50	1.25	7.50	11699.75	2.75	26.50	4017.00	1.00	9.50	11232.25	1.75	20.50	11001.00	1.25	18.25
R_{14}^{r1}	5095.50	1.25	10.25	29818.75	6.75	56.75	8831.75	1.25	13.75	8755.00	1.00	11.50	9144.75	1.00	11.50
R_{15}^{r1}	3061.50	1.00	6.00	7222.25	3.00	17.75	14795.25	4.75	33.75	29072.00	9.25	64.50	4048.25	1.00	7.75
R_{16}^{r1}	3241.00	1.75	8.00	13907.50	2.50	17.75	9906.50	2.00	16.00	26057.00	4.25	34.75	32543.25	7.25	58.50
R_{17}^{r1}	17380.50	4.75	25.75	54830.00	11.00	66.00	16563.00	1.00	11.50	15793.25	1.25	13.25	13025.00	1.25	11.75
R_{18}^{r1}	89928.50	10.50	66.25	37693.25	2.75	23.00	35072.75	1.00	15.00	46624.50	1.00	18.00	47248.50	1.25	19.75
R_{19}^{r1}	11588.25	1.50	9.50	15571.50	2.25	14.75	64515.25	8.50	66.25	46733.75	5.25	41.50	14680.00	1.00	9.75
R_{20}^{r1}	21068.25	5.50	21.00	27233.75	5.00	24.00	12866.00	1.00	8.75	19180.25	1.50	13.25	16906.50	1.25	11.50
R_{21}^{r1}	30317.00	1.25	10.25	55478.00	1.75	17.00	55493.00	2.25	18.50	53078.50	2.25	18.50	153627.50	3.50	40.25
R_{22}^{r1}	62377.00	2.25	16.50	105700.50	1.25	15.50	76303.25	1.00	11.75	127974.75	1.00	15.50	177910.00	1.75	22.25
R_{23}^{r1}	23815.50	2.50	15.00	11747.00	1.25	8.50	32326.00	2.00	16.50	15776.00	1.00	7.75	29713.00	1.50	13.25
R_{24}^{r1}	27348.25	2.25	11.75	59348.25	2.25	17.75	22163.75	1.00	9.00	35701.50	1.00	9.25	34131.75	1.00	10.50
R_{25}^{r1}	62930.50	1.00	10.75	221949.00	2.75	31.25	94512.50	2.00	18.75	85375.00	1.00	11.50	193518.25	3.00	33.00
R_{26}^{r1}	143850.75	1.50	14.25	139858.00	1.00	11.25	383023.00	1.50	23.75	217048.75	1.00	14.25	990130.50	7.25	72.50
AVG	21432.06	3.74	23.94	32584.67	3.21	24.58	36313.89	3.35	28.90	42407.38	5.87	54.76	69349.70	2.35	24.98

Table A.1: Computational results collected for the networks of the sets $S1$ and $S2$ with γ chosen in the interval $r1$.

const	150			225			300			350			400		
test	time	iter	nodes	time	iter	nodes	time	iter	nodes	time	iter	nodes	time	iter	nodes
R_1^{r2}	2324.25	1.25	12.50	3942.75	3.00	23.75	5335.00	4.75	38.25	5042.75	3.25	31.00	14009.00	8.00	83.75
R_2^{r2}	22551.00	11.25	115.50	1692.25	1.25	11.25	3852.25	2.75	23.50	2962.25	2.00	18.50	18425.00	10.25	102.50
R_3^{r2}	2546.75	1.00	11.75	2566.25	1.00	12.00	11068.25	8.25	71.00	4001.75	2.00	21.75	7035.75	5.00	40.50
R_4^{r2}	17679.50	17.25	125.75	5981.50	5.25	38.50	7439.50	3.00	34.25	3367.50	1.50	15.25	2972.25	1.25	13.75
R_5^{r2}	1603.00	1.00	9.00	8446.75	6.00	45.75	31601.25	18.25	159.25	30167.75	15.75	154.50	17380.00	7.00	72.25
R_6^{r2}	3946.75	4.25	25.00	1723.50	1.50	10.25	3002.75	2.00	16.25	2472.75	1.50	13.00	7667.50	4.50	37.50
R_7^{r2}	3109.75	3.50	20.00	3114.00	2.50	18.00	2472.00	1.75	13.00	5428.25	3.25	26.75	4132.25	2.00	17.25
R_8^{r2}	1364.25	1.00	6.75	2380.25	1.75	12.25	49368.50	28.00	220.00	3044.75	1.50	13.50	2693.00	1.25	11.50
R_9^{r2}	4071.50	3.50	20.75	15282.75	10.75	71.25	6983.50	3.75	31.50	2651.25	1.00	9.25	31205.75	12.00	115.25
R_{10}^{r2}	3245.75	2.50	17.75	6386.75	4.00	29.00	8195.00	4.00	34.00	194139.25	77.25	699.25	122182.50	47.50	447.25
R_{11}^{r2}	2046.50	1.75	9.50	4421.00	3.75	21.00	7209.50	4.75	33.00	1443.00	1.25	6.75	8385.75	4.00	29.25
R_{12}^{r2}	854.00	1.00	4.00	7866.25	3.50	20.00	3931.25	1.00	8.00	11610.25	4.25	31.50	25720.50	9.00	75.50
R_{13}^{r2}	3396.75	2.00	11.25	5557.50	1.75	16.25	5997.00	2.00	17.75	44390.25	12.50	119.50	3439.75	1.00	9.75
R_{14}^{r2}	18632.00	1.00	6.50	24717.75	2.25	14.00	49121.00	3.25	30.50	80975.00	5.00	45.25	42100.50	2.00	20.25
R_{15}^{r2}	4555.25	2.00	10.50	3763.25	1.00	7.75	13084.25	3.00	25.25	29121.25	8.00	61.00	9324.75	1.75	17.75
R_{16}^{r2}	4348.75	1.25	7.00	3393.00	1.50	7.00	23672.75	5.50	42.50	14414.00	3.25	24.50	43880.25	8.00	73.50
R_{17}^{r2}	4809.00	1.00	6.50	5132.50	1.00	6.25	13235.25	1.75	14.00	96277.25	11.75	87.75	112737.25	13.25	108.25
R_{18}^{r2}	68975.00	7.25	52.75	38867.25	2.50	21.25	1834837.25	107.25	1037.50	107916.50	5.00	54.75	573450.75	25.75	272.25
R_{19}^{r2}	15530.25	3.00	15.75	4882.25	1.25	6.25	9003.00	1.50	10.25	12587.25	1.00	9.25	47716.50	6.00	44.75
R_{20}^{r2}	12074.50	2.25	11.75	14024.25	2.00	12.50	21332.75	2.25	16.25	20681.50	2.25	17.25	177302.00	16.50	138.75
R_{21}^{r2}	26364.00	1.00	8.50	167090.00	8.25	68.25	34565.75	2.00	15.50	34054.75	1.00	11.00	835755.25	24.00	293.00
R_{22}^{r2}	31085.50	1.00	8.25	297115.25	9.00	70.50	52295.00	1.00	10.25	200978.50	3.25	34.00	158991.75	2.00	22.25
R_{23}^{r2}	18632.00	1.00	6.50	24717.75	2.25	14.00	49121.00	3.25	30.50	80975.00	5.00	45.25	42100.50	2.00	20.25
R_{24}^{r2}	29729.50	2.25	13.00	38604.00	2.50	16.50	46225.25	1.00	13.00	31746.00	1.00	11.25	88379.50	3.00	29.25
R_{25}^{r2}	20693.50	1.00	6.50	136405.00	3.00	23.50	140138.50	2.75	25.25	121192.50	1.25	14.75	266081.50	4.75	46.00
R_{26}^{r2}	111754.25	1.00	10.00	98807.00	1.00	10.00	427551.50	2.25	28.25	143511.00	1.25	11.50	655037.75	3.50	38.50
AVG	16766.28	2.93	21.27	35649.26	3.21	23.35	110024.58	8.50	76.88	49428.93	6.77	61.08	127619.51	8.66	83.88

Table A.2: Computational results collected for the networks of the sets $S1$ and $S2$ with γ chosen in the interval $r2$.

const	150			225			300			350			400		
test	time	iter	nodes	time	iter	nodes	time	iter	nodes	time	iter	nodes	time	iter	nodes
R_1^{r3}	2367.00	1.50	14.00	1107.50	1.50	9.50	8828.25	7.25	58.75	22704.25	18.00	150.75	2535.00	1.00	13.75
R_2^{r3}	9023.50	10.00	60.75	1977.75	1.50	13.50	20537.00	14.75	126.50	2623.75	1.75	16.25	10285.75	4.50	51.50
R_3^{r3}	1432.25	2.00	12.75	1704.00	1.25	10.25	11571.50	9.75	72.25	12426.50	8.50	75.25	2779.00	1.00	12.50
R_4^{r3}	858.25	1.00	6.75	24901.00	22.25	161.00	6085.25	3.50	34.00	1404.50	1.00	8.25	4718.00	2.25	22.75
R_5^{r3}	1458.75	1.25	10.25	64030.25	41.00	390.00	6068.25	4.50	31.25	220623.50	128.00	1182.75	36870.50	16.75	175.25
R_6^{r3}	4313.25	4.75	28.50	1052.75	1.25	8.00	52125.00	32.25	264.25	963.25	1.00	7.25	192009.00	111.75	972.25
R_7^{r3}	2819.25	2.00	14.25	2031.50	1.25	9.50	3516.50	1.75	16.50	16451.25	9.00	79.50	5135.75	2.00	20.75
R_8^{r3}	5584.75	3.25	24.00	5674.50	2.00	18.75	61291.25	35.50	272.25	11028.00	4.75	47.50	4224.50	1.50	16.25
R_9^{r3}	2153.00	1.25	9.50	2271.50	1.50	11.50	1898.50	1.00	8.75	14174.00	6.50	58.50	83307.25	29.50	306.00
R_{10}^{r3}	133944.50	79.00	605.50	16982.25	9.75	71.75	9174.00	4.25	36.25	202029.00	77.75	704.25	3525.00	1.25	12.00
R_{11}^{r3}	3069.25	3.00	14.50	3092.75	2.25	13.25	3159.00	2.25	13.50	3026.25	2.00	14.25	8447.50	5.25	39.75
R_{12}^{r3}	1320.25	1.00	5.25	1388.50	1.00	6.50	6279.25	3.75	26.50	178497.75	94.25	739.75	18636.00	9.75	68.50
R_{13}^{r3}	5405.00	1.50	13.25	11700.75	5.00	36.00	4836.00	1.25	12.00	261482.25	86.25	711.25	35069.00	5.75	62.75
R_{14}^{r3}	60963.75	16.25	120.00	21405.25	2.00	25.00	46718.25	5.75	61.50	34407.75	5.50	48.25	139806.50	16.00	174.25
R_{15}^{r3}	7484.00	2.00	12.50	16329.25	5.25	31.75	30369.25	8.75	68.75	47350.00	10.75	109.50	10407.50	1.75	17.00
R_{16}^{r3}	8196.50	1.00	8.50	14661.25	4.50	25.25	283785.25	58.50	459.75	21924.00	4.50	33.50	8547.50	1.00	9.75
R_{17}^{r3}	18387.50	4.00	25.25	7552.75	1.50	10.00	13157.25	1.75	13.00	17129.75	1.00	12.00	88763.00	7.75	88.75
R_{18}^{r3}	58632.75	6.50	42.25	25712.50	1.75	15.25	2858421.00	161.00	1403.50	1817149.25	86.75	894.50	1446256.75	65.00	669.00
R_{19}^{r3}	24655.25	4.25	22.50	61320.50	10.50	58.75	8402.00	1.00	6.25	9644.25	1.00	8.00	25064.50	3.50	23.00
R_{20}^{r3}	15010.75	1.50	9.50	16083.75	1.75	11.75	62903.25	6.75	50.75	35462.75	3.00	25.00	28597.50	2.75	24.00
R_{21}^{r3}	10954.25	1.00	6.25	35258.50	1.00	9.75	101275.50	4.50	38.00	671126.25	32.75	254.75	1011965.75	38.50	363.25
R_{22}^{r3}	12916.03	1.00	6.07	62695.78	1.09	10.07	153221.10	2.25	25.13	3960738.14	106.44	787.41	192513.54	3.21	27.58
R_{23}^{r3}	858.25	1.00	6.75	24901.00	22.25	161.00	6085.25	3.50	34.00	1404.50	1.00	8.25	4718.00	2.25	22.75
R_{24}^{r3}	18604.50	1.00	6.75	69982.50	5.00	30.50	84765.75	5.50	36.50	75653.75	3.25	31.50	178630.50	7.75	75.75
R_{25}^{r3}	86677.50	1.00	12.25	131008.50	5.00	32.00	145294.25	3.00	28.50	318302.50	6.25	57.00	133082.25	1.50	19.75
R_{26}^{r3}	221271.25	1.00	13.25	67068.00	1.00	8.00	171733.75	1.00	11.25	399327.75	1.00	19.75	1957274.75	13.00	135.00
AVG	27629.28	5.88	42.73	26611.33	5.93	45.71	160057.75	14.81	123.45	321425.19	27.00	234.03	216660.40	13.70	131.69

Table A.3: Computational results collected for the networks of the sets $S1$ and $S2$ with γ chosen in the interval $r3$.

$const$	150			225			300			350			400		
test	time	iter	nodes	time	iter	nodes	time	iter	nodes	time	iter	nodes	time	iter	nodes
c_{51}^{r1}	265.50	1.75	4.75	651.00	1.00	6.50	1115.50	2.25	11.75	11286.75	21.50	137.00	2496.25	5.50	28.50
c_{101}^{r1}	2022.75	1.50	6.00	4571.00	1.75	11.00	10249.00	4.50	26.00	12195.25	5.25	28.75	12788.00	6.75	36.25
r_{51}^{r1}	1981.00	5.00	23.75	3026.50	6.50	34.00	2024.25	3.25	21.75	908.75	1.25	8.50	760.50	1.00	6.50
r_{101}^{r1}	3112.25	2.00	9.50	11967.75	6.50	32.00	6555.75	2.25	21.50	13193.75	6.50	39.75	5300.25	3.75	18.50
rc_{51}^{r1}	17362.75	8.00	50.50	79704.50	28.25	221.00	15260.75	4.50	42.00	9333.00	3.25	24.00	5456.50	3.50	14.50
rc_{101}^{r1}	702.00	1.25	6.00	6088.00	10.25	58.25	1505.25	2.00	15.00	905.00	1.75	9.25	280.75	1.00	3.75
AVG	4241.04	3.25	16.75	17668.13	9.04	60.46	6118.42	3.13	23.00	7970.42	6.58	41.21	4513.71	3.58	18.00
c_{51}^{r2}	405.75	1.50	5.75	635.50	1.75	8.50	604.25	1.00	6.75	973.00	3.00	13.75	22056.00	30.00	227.75
c_{101}^{r2}	3131.50	1.75	9.00	3919.00	2.75	12.00	2366.75	1.50	8.25	25533.25	15.00	76.25	15799.00	8.00	46.75
r_{51}^{r2}	1154.75	3.00	15.00	1368.75	2.50	17.00	9044.25	16.00	99.25	44374.50	51.00	534.50	11512.00	15.00	122.25
r_{101}^{r2}	2667.25	1.50	7.25	24938.75	8.00	59.75	27694.25	10.00	70.25	5780.00	2.25	18.50	491291.50	119.75	1358.25
rc_{51}^{r2}	2929.00	1.25	8.00	96126.25	28.75	276.00	5000.00	1.50	12.00	21703.75	5.75	48.00	3814.25	1.25	8.75
rc_{101}^{r2}	1435.00	3.00	17.50	48398.75	59.75	497.75	1715.75	3.50	18.75	4205.00	3.75	50.00	865.75	1.75	11.25
AVG	1953.88	2.00	10.42	29231.17	17.25	145.17	7737.54	5.58	35.88	17094.92	13.46	123.50	90889.75	29.29	295.83
c_{51}^{r3}	483.75	1.25	4.75	943.75	2.00	10.50	1642.25	3.50	18.75	1956.50	5.00	23.25	12097.75	17.75	136.25
c_{101}^{r3}	2636.25	1.00	5.50	7929.25	3.50	19.00	36894.00	14.50	91.75	6528.75	2.25	17.75	36176.50	19.75	105.00
r_{51}^{r3}	597.00	1.50	7.25	1131.00	2.00	12.00	53956.50	74.75	664.25	47572.25	52.00	545.00	10077.75	12.00	105.25
r_{101}^{r3}	34072.50	19.25	104.75	16575.75	5.50	35.00	15834.00	5.50	42.25	237549.25	68.25	703.50	917873.25	191.50	2008.25
rc_{51}^{r3}	1840.75	1.00	6.00	7441.00	2.75	16.50	5511.00	2.00	14.50	936830.50	168.25	2101.50	25708.75	6.00	67.25
rc_{101}^{r3}	2870.50	6.25	34.50	903829.50	1060.25	9496.50	54264.50	76.00	536.25	7803.75	7.25	98.00	1801029.50	961.00	16201.25
AVG	7083.46	5.04	27.13	156308.38	179.33	1598.25	28017.04	29.38	227.96	206373.50	50.50	581.50	467160.58	201.33	3103.88

Table A.4: Computational results collected for the networks of the set $S3$.

const	150			225			300			350			400		
test	time	iter	nodes	time	iter	nodes	time	iter	nodes	time	iter	nodes	time	iter	nodes
C_1^{r1}	1872.00	4.25	20.00	1813.50	4.25	22.25	702.00	1.50	9.50	1084.25	1.00	8.00	14562.50	19.25	158.50
C_2^{r1}	1973.50	1.75	8.50	3232.75	2.75	14.75	1263.50	1.00	5.50	3669.75	3.00	18.50	2191.75	1.50	10.25
C_3^{r1}	3030.75	1.50	7.50	3517.75	1.00	5.75	14868.25	5.50	35.75	8728.75	4.00	20.00	4298.00	1.25	7.50
C_4^{r1}	7028.75	1.75	8.25	13746.50	2.25	14.00	15142.25	2.75	18.75	11709.25	1.50	12.25	14073.75	1.75	12.75
C_5^{r1}	14405.75	2.00	11.00	21568.75	3.50	19.50	12836.00	1.50	11.00	11852.50	1.00	7.50	25820.50	3.75	24.25
C_6^{r1}	17770.25	2.00	12.25	30599.75	4.00	21.00	18746.75	2.25	13.25	21129.75	2.25	14.00	29893.25	2.75	20.25
C_7^{r1}	17566.25	1.25	6.75	20067.50	1.00	6.50	53133.50	4.00	23.00	97559.00	5.00	37.25	23230.50	1.00	6.25
AVG	9092.46	2.07	10.61	13506.64	2.68	14.82	16670.32	2.64	16.68	22247.61	2.54	16.79	16295.75	4.46	34.25
C_1^{r2}	1313.75	2.50	13.25	1505.50	2.50	15.00	3275.50	5.00	33.75	932.75	13.75	95.00	39675.00	46.50	393.00
C_2^{r2}	3481.00	2.25	12.75	2340.25	1.50	9.00	6079.25	4.75	28.00	7206.50	4.50	30.00	13271.75	7.00	53.50
C_3^{r2}	10307.00	6.00	26.00	6492.50	2.00	14.00	2978.75	1.50	9.25	6664.00	3.25	18.75	7019.25	1.75	13.75
C_4^{r2}	5073.25	1.25	5.75	20697.50	2.75	17.75	54303.50	8.00	60.25	11758.00	1.25	11.00	47993.50	6.50	48.50
C_5^{r2}	22667.50	3.50	15.75	9887.75	1.25	8.25	53698.75	5.25	38.50	20520.25	2.00	16.75	14341.75	1.00	8.50
C_6^{r2}	13425.50	1.25	7.50	35917.50	3.75	23.75	66012.00	5.25	39.50	40641.00	3.75	26.75	89988.25	6.00	58.75
C_7^{r2}	9993.50	1.00	5.00	62312.50	4.00	25.00	39872.25	2.00	14.50	40504.75	2.00	16.75	104077.50	5.50	43.00
AVG	9465.93	2.54	12.29	19879.07	2.54	16.11	32317.14	4.54	31.96	19461.04	4.36	30.71	45195.29	10.61	88.43
C_1^{r3}	1017.75	2.00	10.25	21914.25	42.25	241.00	3123.75	5.50	34.50	990.50	1.75	11.00	19862.25	23.75	198.00
C_2^{r3}	1174.25	1.00	5.50	4820.50	3.25	19.50	11688.50	6.75	45.50	2650.50	1.00	9.00	170613.25	71.25	618.25
C_3^{r3}	3279.00	1.50	7.00	5418.00	1.75	10.50	23241.00	8.50	57.50	3433.50	1.00	6.50	60941.50	15.75	137.50
C_4^{r3}	5613.50	1.00	6.25	72471.75	14.75	76.25	31475.75	6.00	34.75	89981.00	12.00	95.50	18185.75	2.50	20.50
C_5^{r3}	30493.50	4.00	22.25	10457.75	1.25	8.50	76506.50	6.75	53.00	12806.75	1.00	9.50	20052.50	2.25	16.00
C_6^{r3}	16373.00	1.25	6.75	46593.00	5.00	30.00	43739.25	3.25	25.50	117513.00	8.00	69.25	652447.00	47.00	393.50
C_7^{r1}	43729.75	3.75	18.25	17955.00	1.50	8.25	194888.75	12.75	80.50	23743.50	1.00	9.75	26071.75	1.25	11.50
AVG	14525.82	2.07	10.89	25661.46	9.96	56.29	54951.93	7.07	47.32	35874.11	3.68	30.07	138310.57	23.39	199.32

Table A.5: Computational results collected for the networks of the set $S4$.

Chapter 8

Solution Approaches for the Elementary Shortest Path Problem

Luigi Di Puglia Pugliese

Department of Electronics, Informatics and Systems, University of Calabria, 87036, Rende (CS), Italy

Francesca Guerriero

Department of Electronics, Informatics and Systems, University of Calabria, 87036, Rende (CS), Italy

Abstract

In this paper, the elementary shortest path problem is considered. Given a directed graph, containing negative cost cycles, the aim is to find the paths with minimum cost from a source node to each other node, that do not contain repeated nodes. The problem under study is formulated by considering the classical mathematical model of the shortest path in which the sub-tours elimination constraints have to be satisfied. Three different solution strategies are proposed to solve the problem under investigation and their theoretical properties are investigated. The first is a dynamic programming approach, the second method is based on the solution of the k shortest paths problem, where k is considered as a variable. In the last approach, upper bounds are computed and the optimality gap is closed by using a branch and bound scheme.

Keywords: shortest path; negative cost cycles; dynamic programming; k shortest paths; branch and bound method.

8.1 Introduction

The Shortest Path Problem ($\mathcal{SPP}$, for short) is one of the most studied problems in network optimization ([46], [57], [65]). The problem comes up in practice and arises as a subproblem in many network optimization algorithms. Solution approaches for the $\mathcal{SPP}$ have been studied for a long time (e.g., [17], [55], [57], [66], [109], [115]). More recently, some improvements and computational study have been presented in [4], [69], [73]. For a detailed survey on the algorithms for solving the $\mathcal{SPP}$ the reader is referred to [70] and [29].

In its basic formulation, the objective is to determine the minimum cost path through a network from a given origin node to a destination node. Several polynomial time solution approaches have been developed in the scientific literature to address the $\mathcal{SPP}$ (e.g., see [47], [57], [99]). On a network with negative length arcs but with non-negative cost cycles, the best currently known time bound $\mathcal{O}(nm)$ is achieved by the Bellman-Ford-Moore algorithm ([17], [66], [109]), where n and m denote the number of nodes and arcs in the network, respectively. If the arc lengths are non-negative, implementations of Dijkstra's algorithm ([57]) achieve better bounds. In particular, an implementation presented in [68] runs in $\mathcal{O}(m + n \log n)$ time.

A trivial extension of the $\mathcal{SPP}$ is to find the shortest tree rooted at a source node, by considering as destinations all the nodes of the network. This problem is known in the scientific literature as spanning tree problem ($\mathcal{STP}$, for short). A detailed description and a computational study on algorithms for solving the $\mathcal{STP}$ can be found in [15].

An interesting extension of the $\mathcal{SPP}$ is the problem to find the first k shortest paths ($k\mathcal{SPP}$, for short). For each node, a set of k shortest paths have to be determined by considering the first, the second and so on up to the k-th shortest path. This problem arises in many real-life applications, e.g. in telecommunication and transportation fields. In these contexts, alternative solutions should be available when the current best is forbidden for various reason, e.g. link interruption in telecommunication network or street closed in transportation plant. Many solution approaches have been defined to solve the $k\mathcal{SPP}$. The scientific literature takes into account two variants of the $k\mathcal{SPP}$:1) the pure $k\mathcal{SPP}$ and 2)the so-called loopless $k\mathcal{SPP}$. In the first case, a node can appear more than once in the solution paths, whereas, in the latter we are interested in finding the first k shortest paths such that they contain a node only once. A wide number of papers address the $k\mathcal{SPP}$ ([9], [31], [58], [62], [77], [100], [127]). Solution approaches for the loopless $kSPP$ have been defined in [104] and [146]. Dreyfus ([58]) and Yen ([146]) cite several additional papers on this subject going back as far as 1957.

When negative cost cycles are present in the network, the $\mathcal{SPP}$ is not well defined, that is a finite optimal solution does not exist. In this case, the problem is to check whether there exists a simple cycle in this network whose arc costs sum up to a negative number. In the scientific literature this problem is known as Negative Cost Cycle Detection ($\mathcal{NCCD}$, for short) problem.

Approaches to $\mathcal{NCCD}$ can be broadly categorized as relaxation-based [38] or contraction-based [132]. The former approaches are also colled label-correcting methods. The relaxation approach of the Bellman-Ford algorithm is one of the earliest and to date, asymptotically fastest algorithm for $\mathcal{NCCD}$. Depending on the heuristic used to select arcs to be relaxed, a number of relaxation-based approaches are possible ([29], [72]). Recently, in [131] the author introduces a new approach for negative cost cycle detection; the approach is based on exploiting the connections between the $\mathcal{NCCD}$ problem and the problem of checking whether a system of difference constraints is feasible. The author concludes that the proposed procedure is as efficient as the fastest known comparison-based algorithm for this problem.

The problem to find a shortest path in a network with negative cost cycle is still a very challenging problem. To the best of our knowledge, no solution approaches have been defined up to now to solve the elementary shortest path problem ($\mathcal{ESPP}$, for short). It is of course, well known that computing shortest paths in the presence of negative cost cycles is a strongly NP-hard problem [71].

When further constraints on the consumption of resource along the path are introduced, the problem is referred to as the resource constrained elementary shortest path problem ($\mathcal{RCESPP}$, for short). This problem has attracted the attention of many researchers in the last years. Starting to the seminal work of Beasley and Christofides ([16]), a variety of solution approaches have been developed to optimality solve the $\mathcal{RCESPP}$. We cite the work of Feillet et al. ([64]), where the ideas introduced in [16] have been used to define an optimal solution approach and a computational study to assess the performance of the proposed solution strategies has been carried out for the first time.

In the paper of Boland et al. ([22]) and of Righini and Salani ([120]), methods based on a relaxation of the $\mathcal{RCESPP}$ have been proposed. These approaches resemble the idea of Kohl [96]. Briefly, the $\mathcal{RCESPP}$ is solved by considering a relaxed problem in which the constraints on the elementarity are not taken into account. The state-space is iteratively incremented adding the constraint on the nodes that appear more than once in the path until the solution of the relaxed problem is an elementary path. The state-space is augmented by introducing a dummy resource on the node that is repeated along the optimal path of the relaxed problem. The methods proposed in [22] and in [120] turn out to be the most efficient solution approaches.

It is evident that the $\mathcal{ESPP}$ can be viewed as a particular instance of the $\mathcal{RCESPP}$. However, it is worth observing that the solution approaches proposed to solve the latter cannot be directly applied for solving the former. If the best known methods for the $\mathcal{RCESPP}$ are applied to the $\mathcal{ESPP}$, then the constraints on the elementarity are relaxed, thus the relaxation of the $\mathcal{ESPP}$, that is the $\mathcal{SPP}$ is not well defined.

In this paper, we extend the concepts proposed in [22] and in [120] to the general case, that is the $\mathcal{ESPP}$. Other solution strategies are also exploited and an analysis is conducted in order to evaluate the theoretical complexity of the proposed solution approaches.

The paper is organized as follows. In Section 8.2 we give some notations and definitions used in the paper. The proposed solution approaches, along with their theoretical analysis and comparison are presented in Section 8.3, whereas in Section 8.4 we give some conclusions.

8.2 Notations and Definitions

Let $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ be a directed graph where $\mathcal{N}$ is the set of n nodes, whereas $\mathcal{A}$ denotes tha set of $m \leq |\mathcal{N} \times \mathcal{N}|$ arcs. $\mathcal{N}$ contains also the source node s and the destination node d. In the sequel, we assume that $(j, s), (d, j) \notin \mathcal{A}$, $\forall j \in \mathcal{N}$, that is arcs entering node s and leaving node d do not exist in $\mathcal{G}$. A cost c_{ij} is associated with each arc $(i, j) \in \mathcal{A}$. A cycle on node i is defined as a sequence of nodes

$C_i = \{i, j_1, \ldots, j_l, i\}$ and it can be also viewed as a sequence of $l+1$ arcs $\{(i, j_1), \ldots, (j_l, i)\}$ where $l = 1, \ldots, L(C_i)$. The value $L(C_i) = |C_i| - 2$ denotes the number of nodes in the cycle exluding those that appeared twice. A path π_{si} from node s to node i is a sequence of node $\pi_{si} = \{s = i_1, \ldots, i_l = i\}$, $l \geq 1$ and a corresponding sequence of $l-1$ arcs such that the h-th arc in the sequence is $(i_h, i_{h+1}) \in \mathcal{A}$ for $h = 1, \ldots, l-1$. Thus, each path contains at least one arc. Let $M_\pi(j)$ be the multiplicity of node j in path π, that is $M_\pi(j) = |\{v : 0 \leq v \leq |\pi|, i_v = j\}|$. The path π is said to be an *elementary* path if $M_\pi(j) = 1$, for all $j \in \pi$. The cost of a sequence of nodes ND, and thus the corresponding sequence of arc, is defined as $c(DN) = \sum_{(i,j) \in ND} c_{ij}$. A cycle C_i on node $i \in \mathcal{N}$ is said to be a negative cost cycle ($\mathcal{NCC}$) if $c(C_i) < 0$.

Since the directed graph $\mathcal{G}$ is not assumed to be acyclic and the costs are not constrained in sign, thus there may be $\mathcal{NCC}$ in $\mathcal{G}$.

The $\mathcal{ESPP}$, from the source s to the destination d, consists in finding the path π_{sd} such that $c(\pi_{sd})$ is minimized and $M_{\pi_{sd}}(j) = 1,\ \forall j \in \pi_{sd}$.

From a mathematical stand point, the single-source single-destination $\mathcal{ESPP}$ can be formulated by the linear integer programming shown in what follows:

$$\min \sum_{(i,j)\in\mathcal{A}} c_{ij}x_{ij} \tag{8.1}$$

$s.t.$

$$\sum_{\{j:(i,j)\in\mathcal{A}\}} x_{ij} - \sum_{\{j:(j,i)\in\mathcal{A}\}} x_{ji} = 0, \forall i \in \mathcal{N} - \{s,d\} \tag{8.2}$$

$$\sum_{\{j:(s,j)\in\mathcal{A}\}} x_{sj} = 1 \tag{8.3}$$

$$\sum_{\{j:(j,d)\in\mathcal{A}\}} x_{jd} = 1 \tag{8.4}$$

$$\sum_{i\in sN}\sum_{j\in sN} x_{ij} \leq |sN| - 1, \ \ \forall\, sN \subseteq \mathcal{N} \tag{8.5}$$

$$x_{ij} \in \{0,1\} \quad \forall (i,j) \in \mathcal{A} \tag{8.6}$$

Equation (8.2) is the degree constraint for each node of the directed graph excluding the source and the destination, while equations (8.3) and (8.4) ensure that one arc leaves the source node and one arc enters the destination node, respectively. Conditions (8.5) represent the subtour elimination constraints, where sN is a subset of nodes belonging to $\mathcal{N}$. The domain of the decision variable x_{ij}, $\forall(i,j) \in \mathcal{A}$ is defined by (8.6).

It is worth observing that the mathematical model reported above is the same as the classical formulation of the single-source single-destination $\mathcal{SPP}$, if condition (8.5) are removed. These additional constraints guarantee that a finite optimum can be found.

Let y_i be the label that stores the value of the dual price for each node $i \in \mathcal{N}$ and L denote the list of nodes, that have to be processed. We denote as $p \in R^{|\mathcal{N}|}$ the vector of predecessors, that is $p[i] = j$ is the predecessor of node i in the path π_{si} from node s to node i. A set $FS(i)$ of successor nodes is associated with each node $i \in \mathcal{N}$, that is defined as follows: $FS(i) = \{j : (i,j) \in \mathcal{A}\}$.

A general scheme of a label-correcting method for the $\mathcal{ESPP}$ is reported in Algorithm 21.

Algorithm 21 Label Correcting Scheme

```
Step 1 (Initialization phase)
$y_s = 0, y_i = +\infty,\ \forall i \in \mathcal{N} - \{s\}; p[i] = i,\ \forall i \in \mathcal{N}; L = \{s\}$.

Step 2 (Node selection)
Select a node $i$ from the list $L$ and remove it from $L$.

Step 3 (Label extention)
for all $j \in FS(i)$ do
  if $y_j > y_i + c_{ij}$ then
    if $j \notin \pi_{si}$ then
      $y_j = y_i + c_{ij}$;
      $p[j] = i$;
      add node $j$ to $L$ if it does not already belong to it.
    end if
  end if
end for

Step 4 (Termination check)
if $L = \emptyset$ then
  STOP. $y_i,\ \forall i \in \mathcal{N}$ is an upper bound on the optimal cost of the path from
  node $s$ to node $i$.
else
  Go to Step 2.
end if
```

Algorithm 21 is a classical label-correcting method where the elementarity constraints are taken into account. The condition introduced in line 10 ensures that the built paths do not contain repeated nodes and that the algorithm terminates in a finite number of iterations. However, Algorithm 21 cannot be applied to optimality solve the $\mathcal{ESPP}$. This means that at the end of the algorithm, the label $y_i,\ \forall i \in \mathcal{N}$ represents an upper bound on the shortest path from node s to node i. Indeed, the Bellmann's optimality principle is not verified for this problem. Let us assume that π^*_{sd} represents the path with minimum cost and no repeated nodes in a graph with $\mathcal{NCC}s$. We prove that sub-paths of the optimal path are not optimal. This result is formally stated in the following Theorem.

Theorem 8.2.1. *The optimal path π^*_{sd} from node s to node d is composed by sub-paths π_{si} from node s to each other node $i \in \pi_{sd}$ that cannot be optimal for node i.*

Proof. We will prove this theorem by contradiction. Thus, we assume (for the purposes of contradiction) that for π^*_{sd} the following optimality conditions hold:

$$y_z = y_w + c_{wz}, \forall\ (w, z) \in \pi^*_{sd}; \tag{8.7}$$

$$y_v \leq y_u + c_{uv}, \forall\ (u, v) \notin \pi^*_{sd}. \tag{8.8}$$

Let us suppose that a sub-path $\pi^*_{sj} = \pi^*_{si} \cup \pi^*_{ij}$ is part of the optimal solution π^*_{sd}, thus from condition (8.7) we have that $y^*_j = y^*_i + c(\pi^*_{ij})$. Let us suppose that a path $\bar{\pi}_{sj} : i \notin \bar{\pi}_{sj}$ exists to reach node j. Since $\bar{\pi}_{sj}$ does not belong to π^*_{sj}, thus $\bar{y}_j \geq y^*_j$.

Let us suppose to extend path $\bar{\pi}_{sj}$ to node i through path $\bar{\pi}_{ji}$. By the optimality conditions, it follows that $\bar{y}_i = \bar{y}_j + c(\bar{\pi}_{ji}) \geq y^*_i$. If $\pi^*_{ij} \cup \bar{\pi}_{ji} = C_i$ is a $\mathcal{NCC}$, then $\bar{y}_i \geq y^*_i$ is not valid, that is $\bar{y}_i < y^*_i$. This means that $\bar{y}_j + c(\bar{\pi}_{ji}) < y^*_j - c(\pi^*_{ij})$, it follows that $\bar{y}_j - y^*_j < -c(C_i)$. Since $\bar{y}_j - y^*_j \geq 0$, thus $c(C_i) < 0$.

The sub-path π^*_{si} is part of the optimal path π^*_{sj} but it is not optimal for node i. This contradicts the assumption and concludes the proof. □

The theoretical results of Theorem 8.2.1 suggest that it should be necessary to keep more than one path for some node. In addition, if the single-source all-destination version of the $\mathcal{ESPP}$ is considered, the optimal solution is not a spanning tree. Indeed, for some node more than one arc enters it, one for each path that is used to generate optimal path for other nodes.

We remark that the cost of the spanning tree determined by applying Algorithm 21 represents an upper bound on the solution of the single-source all-destination $\mathcal{ESPP}$. This results can be easily derived from Theorem 8.2.1. It is evident that this is also valid for the single-source single-destination version of the problem.

8.3 Proposed Solution Approaches

The solution approaches proposed in this paper are based on the idea to compute for each node the minimum set of paths such that the elementary shortest path is found for all nodes.

Three strategies have been developed. The first one can be viewed as an application of the methods proposed to solve the $\mathcal{RCESPP}$. A multi-dimensional labeling procedure is devised and the dimension of the label is updated dynamically. The second strategy is based on a labeling approach to solve the $k\mathcal{SPP}$. In our case, k is considered as a variable that can take different values for each node. In the last solution approach an upper bound is computed and a branch and bound scheme is used to close the optimality gap.

In what follows, we describe in details the proposed solution approaches.

8.3.1 Dynamic Multi-dimensional Labeling Approach

The approach, presented in this section, uses the concept of *critical node* introduced in [96]. In particular, a node i is said to be critical if there exists a cycle C_i that is a $\mathcal{NCC}$. For each of such a node, a binary value is introduced in order to keep trace about the visiting of the node. This binary variable can be viewed as a resource associated with node i bounded to be less than or equal to one. The concept of node resources was introduced by Beasley and Christofides in [16] for solving the $\mathcal{RCESPP}$.

Let $S = \{i_1, i_2, \ldots, i_{|S|}\}$ be the set of critical nodes and $r[p], p =$

$1, \ldots, |S|$ be the binary variable/resource consumption associated with node $i_p \in S$.

We denote as $y_i = (c(\pi_{si}), r_i)$ the label representing the cost and the resource consumption of the path π_{si} from node s to node i. The resource consumption r_i has the following form: $r_i[p] = 1,\ \forall p : i_p \in \pi_{si}$; $r_i[p] = 0,\ \forall p : i_p \notin \pi_{si}$. In other words, the vector r_i keeps trace of the nodes $i \in S$ that are visited along the path π_{si}.

Definition 8.3.1. *Let $y_i^1 = (c(\pi_{si}^1), r_i^1)$ and $y_i^2 = (c(\pi_{si}^2), r_i^2)$, be two labels associated with node i. The label y_i^2 is dominated by the label y_i^1 if $c(\pi_{si}^1) \leq c(\pi_{si}^2)$, $r_i^1[p] \leq r_i^2[p], p = 1, \ldots, |S|$ and at least one inequality is strict.*

Definition 8.3.2. *A label $y_i = (c(\pi_{si}), r_i)$ associated with node i is said to be efficient if does not exist a label $\bar{y}_i$ that dominates it.*

Definition 8.3.3. *A label $y_i = (c(\pi_{si}), r_i)$ associated with node i is said to be feasible if $r_i[p] \leq 1,\ \forall p = 1, \ldots, |S|$.*

With each node $i \in \mathcal{N}$ is associated a set $D(i)$ that stores all the labels that are efficient, i.e. Pareto optimal. The procedure defined by Desrochers in [50] can be applied to solve the $\mathcal{ESPP}$. It is worth observing that the Desrochers' algorithm is an extension of Algorithm 21 without line 10, to the constrained case. In Algorithm 22 we report the steps of the label-correcting method extended to the constrained case.

It is important to point out that it is possible to devise a label-setting algorithm by slightly modifying Algorithm 22. Indeed, the set L stores all the efficient and feasible labels and it is initialized by the label y_s. At Step 2, from the list L the lexicografically minimal label is selected. This label is used to generate feasible and efficient labels to each successor node. This procedure is the same as the one proposed in [22] called general label-setting algorithm.

The drawback of the procedure is that the set S of critical nodes must be known in advance. This means that a $\mathcal{NCCD}$ problem has to

Algorithm 22 Label Correcting Scheme for the $\mathcal{ESPP}$

```
1:  Step 1 (Initialization phase)
2:  $y_s^1 = (0,0),\ D(s) = \{y_s^1\};\ L = \{s\}$.
3:
4:  Step 2 (Node selection)
5:  Select a node $i$ from the list $L$ and remove it from $L$.
6:
7:  Step 3 (Label extention)
8:  for all $y_i^\xi \in D(i)$ do
9:      for all $j \in FS(i)$ do
10:         Set: $c(\bar{\pi}_{sj}) = c(\pi_{si}^\xi) + c_{ij}$; $\bar{r}_j[p] = r_i^\xi[p], p = 1, \dots, |S|$; if $j \in S$ then $\bar{r}_j[\hat{p}]++, i_{\hat{p}} = j$.
11:         if $\bar{r}_i[p] \leq 1,\ p = 1, \dots, |S|$ then
12:             if $\bar{y}_j = (c(\bar{\pi}_{sj}), \bar{r}_j)$ is not dominated by any label in $D(j)$ then
13:                 $D(j) = D(j) \cup \{\bar{y}_j\}$;
14:                 remove from $D(j)$ all the labels that are dominated by $\bar{y}_j$;
15:                 add node $j$ to $L$ if it does not already belong to it.
16:             end if
17:         end if
18:     end for
19: end for
20:
21: Step 4 (Termination check)
22: if $L = \emptyset$ then
23:     STOP. The label $y_i$ with the smollest cost among those belonging to $D(i)$ is associated with the optimal elementary path from $s$ to $i$, $\forall i \in N$.
24: else
25:     Go to Step 2.
26: end if
```

be solved in order to detect all the $\mathcal{NCC}s$ C_{i_s}. All nodes i_s have to be marked as critical nodes. It is important to point out that both the relaxation-based and contraction-based methods are able to solve the shortest path problem if $\mathcal{NCC}s$ are not present. A check is invoked in order to detect the $\mathcal{NCC}$. The procedures are not able to return all the $\mathcal{NCC}s$, rather they detect if at least one $\mathcal{NCC}$ exists, thus this type of algorithm cannot be used for our scope. In addition, also the new solution approach for the $\mathcal{NCCD}$ problem proposed in [131] is not able to determine the entire set of $\mathcal{NCC}$.

If the set S is not known in advance, then it can be possible to devise an incrementally strategy by following the idea of Kohl ([96]). In particular, starting from $S = \emptyset$, the label-setting algorithm is re-executed. The search is stopped when a $\mathcal{NCC}$ C_i is detected. The set S is incremented by adding node i and the label-setting algorithm is rerun. The procedure terminates when no $\mathcal{NCC}$ is detected. This approach is similar to the general state-space augmenting algorithm proposed in [22]. In Algorithm 23 we describe the steps of the aforementioned procedure.

Algorithm 23 Dynamic Multi-dimentional Labeling Approach

```
1:  **Step 1** *(Initialization phase)*
2:  Set: $\zeta = 0$, $S^\zeta = \emptyset$
3:
4:  **Step 2** *(Run the truncated label-setting algorithm)*
5:  Run Algorithm 24 with critical node restricted to $S^\zeta$.
6:
7:  **Step 3** *(Cycle detection)*
8:  if $C_i$ is detected then
9:      $S^{\zeta+1} = S^\zeta \cup \{i\}$;
10:     $\zeta + +$;
11:     Go to **Step 2**.
12: else
13:     STOP. The label $y_i$ with the smallest cost among those belonging to $D(i)$ is
        associated with the optimal elementary path from $s$ to $i$, $\forall i \in N$.
14: end if
```

It is worth observing that at the first iteration of Algorithm 23, since the set S^0 is an empty set, the multi-dimensional labels contain only the cost, that is $y_i = c(\pi_{si})$ and $|D(i)| = 1$ $\forall i \in \mathcal{N}$. In other words, when $\zeta = 0$ we have an instance of the $\mathcal{SPP}$. The truncated label-setting algorithm is reported in Algorithm 24.

It is a common agree that a bi-directional search can improve the performance of the mono-directional counterpart when the $\mathcal{SPP}$ is solved. The search from both node s and node d were extended to the

Algorithm 24 Truncated label-setting algorithm with S^ζ

```
1: Step 1 (Initialization phase)
2: $y_s^1 = (0, 0),\ D(s) = \{y_s^1\}$; $L = \{y_s^1\}$.
3:
4: Step 2 (Label selection)
5: Select the lexicografically minimal label $y_i^\xi$ from the list $L$ and remove it from $L$.
6:
7: Step 3 (Label extension)
8: for all $j \in FS(i)$ do
9:     Set: $c(\bar{\pi}_{sj}) = c(\pi_{si}^\xi) + c_{ij}$; $\bar{r}_j[p] = r_i^\xi[p], p = 1, \ldots, |S^\zeta|$; if $j \in S^\zeta$ then $\bar{r}_j[\hat{p}] + +, i_{\hat{p}} = j$.
10:    if $j \notin S^\zeta$ and $j \in \pi_{si}^\xi$ and $c(\bar{\pi}_{sj}) < c(\pi_{sj}^\xi)$ then
11:        STOP. A $\mathcal{NCC}$ $C_j$ is detected.
12:    else
13:        if $\bar{r}_i[p] \leq 1,\ p = 1, \ldots, |S^\zeta|$ then
14:            if $\bar{y}_j = (c(\bar{\pi}_{sj}), \bar{r}_j)$ is not dominated by any label in $D(j)$ then
15:                $D(j) = D(j) \cup \{\bar{y}_j\}$, $L = L \cup \{\bar{y}_j\}$;
16:                remove from $D(j)$ and $L$ all the labels that are dominated by $\bar{y}_j$.
17:            end if
18:        end if
19:    end if
20: end for
21:
22: Step 4 (Termination check)
23: if $L = \emptyset$ then
24:     STOP. The label $y_i$ with the smallest cost among those belonging to $D(i)$ is associated with the optimal elementary path from $s$ to $i$, $\forall i \in N$.
25: else
26:     Go to Step 2.
27: end if
```

$\mathcal{SPP}$ with resource constraints in [119]. The authors used the label-correcting methods of Feillet ([64]) to perform the mono-directional serch from both s and d. When no more nodes have to be considered, the paths from node s to node d are computed by joining the forward paths from node s and the backward paths from node d. Both forward and backward labels are extended only if the consumption of resource is less than an half of the available critical resource. This procedure is called by the authors as bounded bi-directional search.

In our case the resources model the elementarity constraints and the corresponding available amount of resource is exactly one. It is evident that the bounded bi-directional search turn out to be less efficient than the mono-directional counterpart. Indeed, the bounding strategy is not able to stop the extension of the label, thus the bi-directional search generates a twice number of labels respect to that computed with a mono-directional search.

In addition, our scope is to determine the elementary path from the source s to all the remaining nodes. If the bi-directional search is used, then in the join phase it is necessary to consider the generation of all the paths from node s to all other nodes of the network.

Complexity analysis

As far as the complexity analysis of the proposed state-space augmenting algorithm for the $\mathcal{ESPP}$, let $\tilde{C}$ be the number of $\mathcal{NCC}s$ that are included in the graph $\mathcal{G}$ and D be the set of efficient and feasible labels with the highest cardinality. The complexity of Algorithm 23, in the worst case, is $\mathcal{O}(n^2|D|^2\sum_{c=1}^{\tilde{C}} c)$.

This results is formally stated in the following lemma.

Lemma 8.3.4. *In the worst case, the complexity of the proposed approach is* $\mathcal{O}(n^2|D|^2\sum_{c=1}^{\tilde{C}} c)$.

Proof. The operations executed in lines 9 - 19 of Algorithm 24 are $|D||S|$, that is the test of dominance. Since the FS can contain at most $n-1$ elements, the total number of operations executed in lines 8 - 20 is $n|D||S|$. Since the forloop of line 8 is invoked $n|D|$ times, that is the total number of generated labels, thus Algorithm 24 takes $\mathcal{O}(n^2|D|^2|S|)$ operations. The number of iterations executed by Algorithm 23 is exactly $\tilde{C}$. This means that, starting from $S = \emptyset$, the set of critical nodes is incremented by one unit at each iteration. In other words, the operations executed by Algorithm 23 are $n^2|D|^2 1 + n^2|D|^2 2 + \ldots + n^2|D|^2 \tilde{C}$. Consequently, the complexity of Algorithm 23 in the worst case is $\mathcal{O}(n^2|D|^2 \sum_{c=1}^{\tilde{C}} c)$ $\square$

8.3.2 Labeling Approach based on the k Shortest Path Method

In this section a labeling approach is presented for solving a dynamic $k\mathcal{SPP}$. The devised procedure is able to solve the $\mathcal{ESPP}$ and it works as described in what follows.

Let K_i be the number of best paths from node s to node i. The set Π_i contains the K_i paths ordered by increasing cost, that is $c(\pi_{si}^k) < c(\pi_{si}^{k+1})$, $\forall k = 1, \ldots, K_i - 1$. In other words, $\Pi_i[k] = \pi_{si}^k$ is the k-st shortest path from node s to node i. With each node i is associated a vector $y_i \in R^{K_i}$ that stores the costs of the paths belonging to the set Π_i, that is $y_i[k]$ represents the cost of path $\pi_{si}^k \in \Pi_i$.

Also in this case we use a truncated labeling algorithm. Every time a cycle C_i is detected, the number of paths that have to be found for the node $j_{L(C_i)} \in C_i$ is increased, that is $K_{j_{L(C_i)}} = K_{j_{L(C_i)}} + 1$ and the truncated labeling algorithm is execute again. When the truncated algorithm does not detect a cycle, $y_i[1]$ is the cost of the optimal elementary shortest path.

The steps of the proposed algorithm are depicted in Algorithm 25.

In order to obtain a finite optimal solution, an iterative scheme is devised. The proposed algorithm iteratively run Algorithm 25 after coherently updated the set K_i, $\forall i \in \mathcal{N}$. The dynamic labeling solution strategy is reported in Algorithm 26.

Algorithm 25 Truncated labeling algorithm for dynamic $k\mathcal{SPP}$

```
Step 1 (Initialization phase)
Set: $y_s[1] = 0$; $y_i[k] = +\infty, k = i, \ldots, K_i$; $\Pi_s = \{\pi^1_{ss}\}, \pi^1_{ss} = \{s\}$; $\Pi_i = \emptyset, \forall i \in \mathcal{N} - \{s\}$; $L = \{s\}$.

Step 2 (Node selection)
Select a node $i$ from $L$ and delete it from $L$.

Step 3 (Label extension)
for all $j \in FS(i)$ do
    if $\forall k : j \in \pi^k_{si}, y^k_i + c_{ij} < y^1_j, k = 1, \ldots, K_i$ AND $\nexists \bar{k} : j \notin \pi^{\bar{k}}_{si}$ then
        STOP. A cycle $C_j$ is detected.
    else
        for all $\bar{k} : j \notin \pi^{\bar{k}}_{si}$ do
            for all $\xi = 1, \ldots, K_j$ do
                if $y_i[\bar{k}] + c_{ij} < y_j[\xi]$ then
                    $y_j[\delta + 1] = y_j[\delta], \delta = \xi, \ldots, K_j$;
                    $\pi^{\delta+1}_{sj} = \pi^{\delta}_{sj}, \delta = \xi, \ldots, K_j$;
                    $y_j[\xi] = y_i[\bar{k}] + c_{ij}$;
                    $\pi^{\xi}_{sj} = \pi^{\bar{k}}_{si} \cup \{j\}$;
                    add node $j$ to $L$ if it does not already belong to it.
                end if
            end for
        end for
    end if
end for

Step 4 (termination check)
if $L = \emptyset$ then
    STOP. $y_i[1]$ is the cost of the optimal elementary path $\pi^1_{si}$.
else
    Go to Step 2.
end if
```

Complexity analysis

Let K be the highest number of paths that have to be found. The complexity of Algorithm 26 is derived in the following lemma.

Lemma 8.3.5. *In the worst case, the complexity of Algorithm 26 is*

Algorithm 26 Dymanic labeling approach

```
1: Step 1 (Initialization phase)
2: Set: $\zeta = 0$; $K_s = 1$; $K_i^{\zeta} = 1 \ \forall i \in \mathcal{N} - \{s\}$.
3:
4: Step 2 (Run the truncated labeling algorithm)
5: Run Algorithm 25 with $K_i^{\zeta}, i \in \mathcal{N}$
6:
7: Step 3 (Cycle detection)
8: if a cycle $C_i$ is detected then
9:     $K_{j_{L(C_i)}}^{\zeta+1} ++$, $K_i^{\zeta+1} = K_i^{\zeta} \ \forall i \in \mathcal{N} - \{j_{L(C_i)}\}$;
10:    $\zeta ++$;
11:    Go to Step 2.
12: else
13:    STOP. $y_i[1]$ is the cost of the optimal elementary path $\pi_{si}^1$.
14: end if
```

$\mathcal{O}(n^2 \sum_{c=1}^{\tilde{C}} c^5)$.

Proof. The operations executed in lines 14 - 20 are K^2. Since the forloops of line 12 and 13 take K^2, thus the number of iterations in lines 12 - 22 is K^4. The FS contains at most $n - 1$ elements. The forloop of line 8 is invoked nK times. Consequently, Algorithm 25 takes $\mathcal{O}(n^2K^5)$. The iterations executed by Algorithm 26 are exactly $\tilde{C}$. This means that the proposed approach executes $n^2 1^5 + n^2 2^5 + \ldots + n^2 \tilde{C}^5$ operations, in other words the complexity of Algorithm 26 is $\mathcal{O}(n^2 \sum_{c=1}^{\tilde{C}} c^5)$. □

8.3.3 Branch and Bound

The proposed branch-and-bound (B&B) solution approach relies on the detection of a $\mathcal{NCC}$. Each time the truncated labeling method, that is Algorithm 25 with $K_i = 1$, $\forall i \in \mathcal{N}$, is stopped because a cycle is found, a number of sub-problems are generated in order to exclude the detected $\mathcal{NCC}$ from the related sub-graph.

Branching rule

Let $\mathcal{G}(\mathcal{N},\mathcal{A})$ be a graph associated with a generic node of the B&B tree. The truncated labeling approach is run on $\mathcal{G}(\mathcal{N},\mathcal{A})$. If a $\mathcal{NCC}$ C_i is detected, then $\delta = |C_i| - 1$ sub-graphs are generated starting from $\mathcal{G}(\mathcal{N},\mathcal{A})$ by removing the $\delta - st$ arc of C_i. In other words, the graph $\mathcal{G}^{\delta}(\mathcal{N},\mathcal{A}^{\delta})$, $\delta = 1,\ldots,|C_i|-1$ are generated where $\mathcal{A}^{\delta} = \mathcal{A} - \{(u_\delta, v_\delta)\}$, with $(u_\delta, v_\delta) \in C_i$.

Since at least one arc of the detected $\mathcal{NCC}$ C_i is removed, the related sub-graph does not contain this cycle. When the truncated labeling algorithm is run on this sub-graph, if a $\mathcal{NCC}$ $\bar{C}_i$, for some node $i \in \mathcal{N}$ is detected, we are sure that $C_i \neq \bar{C}_i$. This means that at a certain level σ of the B&B tree exactly σ $\mathcal{NCC}s$ have been detected. Consequently, in all the sub-graphs at level σ, exactly σ $\mathcal{NCC}s$ are removed.

Algorithm

Let $y^{\zeta} \in R^{|\mathcal{N}|}$ be the vector that stores the best solutions for each node $i \in \mathcal{N}$, determined so far, and $y^* \in R^{|\mathcal{N}|}$ be the vector of the optimal solutions for each node $i \in \mathcal{N}$. The value ζ is the iterations counter. At the last iteration $\bar{\zeta}$, $y^{\bar{\zeta}} = y^*$, that is $y^*[i]$ is the cost of the optimal path from node s to node i. Let $y_{(\mathcal{G})}[i]$ be the cost of the path from node s to node i obtained applying the truncated labeling algorithm on graph $\mathcal{G}$. If a $\mathcal{NCC}$ C_j is detected when the truncated labeling algorithm is executed on $\mathcal{G}$, then we have that $y_{(\mathcal{G})}[i]$ is a valid upper bound for all nodes $i \in \bar{\mathcal{N}}$, where $\bar{\mathcal{N}} = \{u \in \mathcal{N} : v \notin \pi_{su},\ \forall v \in C_j\}$. This means that $y_{(\mathcal{G})}[i]$, $\forall i \in \bar{\mathcal{N}}$ can be used to initialize the cost of node i for the sub-graph $\mathcal{G}^{\delta}$. In other words, $y_{\mathcal{G}^{\delta}}[i] = y_{\mathcal{G}}[i]$, $\forall i \in \bar{N}$. In addition, if $y_{\mathcal{G}}[i] < y^{\zeta}[i]$ for some $i \in \bar{\mathcal{N}}$, then $y^{\zeta}[i] = y_{\mathcal{G}}[i]$. Let L be the list containing the sub-problems/graphs that have to be solved by applying the truncated labeling algorithm. The steps of the proposed B&B approach are reported in Algorithm 27.

Algorithm 27 B&B algorithm

```
1: Step 1 (Initialization phase)
2: Set: $\zeta = 0$; $y^{\zeta}[s] = 0, y^{\zeta}[i] = +\infty,\ \forall i \in \mathcal{N} - \{s\}$; $L^0 = \{\mathcal{G}\}, \mathcal{G} = (\mathcal{N}, \mathcal{A}), y_G[s] = 0, y_G[i] = +\infty,\ \forall i \in \mathcal{N}$.
3:
4: Step 2 (Selection phase)
5: Select a graph $\bar{\mathcal{G}} = (\mathcal{N}, \bar{\mathcal{A}})$ from $L^{\zeta}$ and remove it from $L^{\zeta}$.
6:
7: Step 3 (Branching phase)
8: Let apply the truncated labeling algorithm to $\bar{\mathcal{G}}$.
9: if a $\mathcal{NCC}$ $C_j$ is detected then
10:    Let generate $\delta = |C_j| - 1$ sub-graph $\mathcal{G}^{\delta} = (\mathcal{N}, \bar{\mathcal{A}} - \{(u_{\delta}, v_{\delta})\})$ and add them to $L^{\zeta}$;
11:    set: $y_{\mathcal{G}^{\delta}}[i] = y_{\bar{\mathcal{G}}}[i],\ \forall i \in \bar{\mathcal{N}}$.
12:    if $y_{\bar{\mathcal{G}}}[i] < y^{\zeta}[i]$ for some $i \in \bar{\mathcal{N}}$ then
13:        $y^{\zeta}[i] = y_{\bar{\mathcal{G}}}[i]$.
14:    end if
15:    $y^{\zeta+1} = y^{\zeta}$
16:    $L^{\zeta+1} = L^{\zeta}$
17:    $\zeta + +$;
18:    go to Step 4.
19: else
20:    if $y_{\bar{\mathcal{G}}}[i] < y^{\zeta}[i]$ for some $i \in \mathcal{N}$ then
21:        $y^{\zeta}[i] = y_{\bar{\mathcal{G}}}[i]$.
22:    end if
23:    $y^{\zeta+1} = y^{\zeta}$
24:    $L^{\zeta+1} = L^{\zeta}$
25:    $\zeta + +$;
26:    Go to Step 4
27: end if
28:
29: Step 4 (Termination check)
30: if $L^{\zeta} = \emptyset$ then
31:    STOP. $y^{\zeta-1}[i]$ is the cost of the optimal path from node $s$ to node $i$.
32: else
33:    Go to Step 2.
34: end if
```

Complexity analysis

Let M be the maximum cardinality among those of the $\mathcal{NCC}s$ in the graph. We know that the number of $\mathcal{NCC}s$ in the graph is $\tilde{C}$. In addition, for each level σ of the B&B tree exactly σ $\mathcal{NCC}s$ have been removed from the graphs at level σ. Since the number of $\mathcal{NCC}s$ in the original graph is $\tilde{C}$, thus the levels of the B&B tree are exactly $\tilde{C}$. The complexity of the B&B approach is $\mathcal{O}(n^2M^{\tilde{C}})$. This result is formally stated in the following lemma.

Lemma 8.3.6. *In the worst case, the complexity of the proposed B&B methods for the $\mathcal{ESPP}$ is $\mathcal{O}(n^2M^{\tilde{C}})$.*

Proof. The original graph associated with the first node of the B&B tree is solved and a $\mathcal{NCC}$ is detected. The branching procedure is applied and $\delta = M - 1$ sub-graphs are generated. When each of this sub-graph is solved, another $\mathcal{NCC}$ is detected and $M - 1$ graphs of the second level are generated. This means that the number of graphs that have to be solved is $(M - 1)^{\tilde{C}}$. Since the complexity to solve each graph with Algorithm 25 where $K = 1$ takes $\mathcal{O}(n^2)$ (see lemma 8.3.5), thus the worst case complexity of the proposed B&B method is $\mathcal{O}(n^2M^{\tilde{C}})$. $\square$

8.3.4 Theoretical comparison

As far as the theoretical comparison among the proposed solution approaches is concerned, let us consider the worst case complexity. We remark that the complexity of the dynamic multi-dimensional labeling approach, $\mathcal{DMLA}$ for short (Algorithm 23) is $\mathcal{O}(n^2|D|^2\sum_{c=1}^{\tilde{C}} c)$, whereas, $\mathcal{O}(n^2\sum_{c=1}^{\tilde{C}} c^5)$ is the complexity of the dynamic labeling approach ($\mathcal{DLA}$, for short) that is Algorithm 26 and the complexity of the B&B approach is $\mathcal{O}(n^2M^{\tilde{C}})$.

$\mathcal{DMLA}$ is worse than $\mathcal{DLA}$, that is $\mathcal{O}(n^2|D|^2\sum_{c=1}^{\tilde{C}} c) > \mathcal{O}(n^2\sum_{c=1}^{\tilde{C}} c^5)$ if $|D| > \frac{\sqrt{\sum_{c=1}^{\tilde{C}} c^5}}{\sqrt{\sum_{c=1}^{\tilde{C}} c}}$. Whereas, $\mathcal{DMLA}$ is worse than the B&B approach if $\tilde{C} < \log_M |D|^2\sum_{c=1}^{\tilde{C}} c$. Regarding the comparison between $\mathcal{DLA}$

and the B&B, the latter is better than the former if $\tilde{C} < \log_M \sum_{c=1}^{\tilde{C}} c^5$

8.4 Conclusions

In this paper we have investigated the shortest path problem in presence of negative cost cycles. This study is the first attempt to provide resolution methods for the shortest path problem with negative cost cycles. The scientific literature provides strategies that are able to check if a negative cost cycle exists but no works consider the resolution of the elementary shortest path problem.

Three different strategies have been devised to optimally solve the problem under investigation. The main idea behind the proposed solution approach is to compute, for each node, the minimal number of paths such that the shortest paths from the source node to all others nodes are determined. In addition, all the methods dynamically increase the number of paths that have to be found for some node. The main difference among the proposed approaches is related to the way in which the number of paths is incremented. In the first method, the number paths that have to be found is determined by considering a dummy node resource that keeps trace about the visiting at the node. This results in a constrained multi-objective shortest path in which the node associated with the dummy resource can be visited only once along the paths. The second proposed approach is based on the idea behind the k shortest path methods. In particular, the value of k is different per node and it is incremented each time a further path that pass through such a node is needed in order to avoid the negative cost cycle. The last strategy increases the number of path by considering a branching rule based on cycle elimination.

The theoretical complexity of the proposed solution approaches in the worst case is derived and a theoretical comparison is also made.

It is worth mentioning that an empirical comparison among the defined approaches is necessary. In addition, it is important to evaluate the behaviour of the proposed methods in terms of computational effort and memory occupancy. These represent the subjects of current investigation.

Part III

Conclusions

Chapter 9

Conclusions

In this thesis we have addressed the constrained shortest path problem and some variants of its. This work summarize the main results achieved in the three years of the Ph.D. program. In particular, innovative models and methods have been defined for several instances of the constrained shortest path. The considered problems are listed in what follows:

- the resource constrained shortest path problem ($\mathcal{RCSPP}$, for short);
- the shortest path problem with forbidden paths ($\mathcal{SPPFP}$, for short);
- the elementary shortest path problem with forbideen paths ($\mathcal{ESPFP}$, for short);
- the resource constrained elementary shortest path problem ($\mathcal{RCESPP}$, for short);
- the linear franctional elementary shortest path problem with time windows ($\mathcal{LFESPPTW}$, for short);
- the multi-criteria path problem with multiple metrics and soft constraints ($\mathcal{MPPSC}$, for short);

- the elementary shortest path problem ($\mathcal{ESPP}$, for short).

Innovative methods have been developed, designed and implemented. The obtained theoretical results have been validated by an extensive and exaustive experimental phase in order to asses the behaviour of the porposed solution approaches in terms of both the computatiuonal effort, the memory occupancy and the robustness.

Regarding the $\mathcal{RCSPP}$, an innovative method based on the concept of reference point have been defined. The proposed solution approach has been computationally compared with the best known state of the art algorithms. In addition, new models and methods for computing upper and lower bound have been introduced. The results are very encouraging. Indeed, the poposed resolution strategy is competitive with the considered algorithms taken from the literature and the the upper bounds obtained with the proposed procedure improve the previous results. In addition, the lower bounds abtained with the proposed model are very close to the optimal solution and in most cases the model provides the optimal one.

Solution methodologies to address the $\mathcal{SPPFP}$ have been proposed. The developed algorithms are based on the paradigm of the dynamic programming optimization and can be viewed as modified versions of the Desrochers' algorithm ([50]), for the constrained shortest path. A pruning strategy has been defined with the goal to speed-up the search of the optimal solution. With this aim, upper bounds have been determined by developing a well-tailored heuristic procedure. In addition, we have considered both the node and state extension rule and two types of selection strategies. Indeed, a Dijkstra-like and the Bellman-Ford rules have been implemented to select, at each iteration, the node/state to be processed. The solution approaches have been evaluated numerically on several sets of networks. A comparison with the state-of-art algorithm to solve the problem under consideration and with Desrochers' approach has also been made. The proposed solution approach is very efficient in solving the shortest path problem with a forbidden path and outperforms the polinomial algorithms which appeared quite recently in scientific literature to address the problem at hand.

Regarding the $\mathcal{ESPFP}$, the problem has been formulated as a specific instance of the resource constrained shortest path problem. $B\&B$ and dynamic programming based solution approaches have been defined and implemented. Different versions of the two types of solution approaches have been developed. Regarding to the $B\&B$ method, a naive and two enhanced versions have been defined. Both node and label selection versions of a dynamic programming based algorithm have been considered. In addition, several rules have been implemented to select, at each iteration, the label/node to be processed. An extensive computational study has been carried out on a variety of network instances with the goal of assessing the behavior of the proposed solution procedures. The collected numerical results underline that the performance of the proposed solution approaches is influenced by both the number of the additional constraints and the dimension of the problems that have to be solved. In conclusion, the $B\&B$ strategies developed to solve the elementary shortest path problem with forbidden paths could be competitive with the dynamic programming solution approaches only if the number of violated constrains is very limited respect to the total number of forbidden paths and when the number of nodes and the number of arcs of the considered problem is not too high, i.e., networks with a number of nodes and a number of arcs less than or equal to 400 and to 5000, respectively. In addition, when the number of violated constrains increases, the computational cost of the $B\&B$ methods increases dramatically and they behave very poorly. On the contrary, the dynamic programming approaches seem to be very effective in solving the problem under study.

We have presented multi-dimensional labelling approaches to address the $\mathcal{LFESPPTW}$, defined by using different label selection and node selection strategies, on the basis of which the most promising node/label is selected at each iteration. In order to define efficient solution approaches, a bi-directional search strategy and some dominance rules, well tailored to the problem at hand, have also been exploited. In order to assess the behaviour of the proposed approaches, extensive computational experiments have been carried out on a set of randomly generated networks, with varying size and density. The label selection methods have been compared with the state of art approach to address the problem under study, i.e. the label correcting method in which the candidate list of labels is accessed by a FIFO policy proposed by Roan and Lee in 2003 [122].

For the $\mathcal{RCESPP}$, we have conducted a computational analysis of the two best known algorithms. In addition, we have discussed about the similarity of the two algorithms and we have highlighted the differences related to critical aspects in terms of efficiency. For the computational studies, we have considered the instances proposed by the authors for testing their algorithm. Thus, we have unified the computational setting and the comparison between the two algorithms have been done on the same data-set. In this way, we have studied the behaviour of each algorithm not only on the test problems considered originally by the authors, but also on the instances considered by the others. The computational results suggest that no algorithm dominates the other. In addition, the performance of one of the considered algorithms has been improved by considering some strategies adopted in the other.

Regarding the $\mathcal{MPPSC}$, a branch-and-bound based solution approach has been devised. The computational phase has been carried out by considering different scenarious. Indeed, several types of networks and different possible desiders of the decisor maker have been considered. The experimental results suggest that the performance of the proposed strategy is closely related to the width of the range, the higher the width, the higher the computational effort. No relationship has been observed beetwen the behaviour of the proposed method and the value of the middle of the ranges.

Three different strategies have been devised to optimally solve the $\mathcal{ESPP}$. The first method is based on a dynamic programming framework. This results in a constrained multi-objective shortest path in which the constraints are dummy node resources. In this way, the node associated with the dummy resource can be visited only once along the paths. The second proposed approach is based on the idea behind the k shortest path methods. In particular, the value of k is different per node and it is incremented each time a further path that pass through such a node is needed in order to avoid the negative cost cycle. The last strategy increases the number of path by considering a branching rule based on cycle elimination. A theoretical analysis has been conducted and a comparison among the defined solution approaches has also been made.

Bibliography

[1] J.I. Agbinya. Qos functions and theorems for moving wireless networks. In *International Conference on Information Technology and Application*, 2005.

[2] A.V. Aho and M.J. Corasick. Efficient string matching: An aid to bibliographic search. *Journal of the ACM*, 18(6):333–340, 1975.

[3] R.K. Ahuja. Minimum cost-reliability ratio path problem. *Comput. Oper. Res.*, 15:83–89, 1988.

[4] R.K. Ahuja, K. Mehlhorn, J.B. Orlin, and R.E. Tarjan. Faster algorithms for the shortest path problem. Technical Report CS-TR-154-88, Departement of Computer Science, Princeton University, 1988.

[5] Y.P. Aneja, V. Aggarwal, and K.P.K. Nair. Shortest chain subject to side constraints. *Networks*, 13:295–302, 1983.

[6] C. H. Antunes, J. Craveirinha, J. Climaco, and J. Barrico. Multiple objective routing in integrated communication networks. In P. Key and D. Smith, editors, *ITC-16 Teletraffic engineering in a competitive world*, pages 1291 – 1300, North Holland, 1999. Elsevier.

[7] P. Avella, M. Boccia, and A. Sforza. Resource constrained shortest path problems in path planning for fleet management. *Journal of Mathematical Modelling and Algorithms*, 3:1–17, 2004.

[8] A. Awajan, K. Al-Begain, and P. Thomas. Quality of service routing for real-time applications using multiple criteria decision making methods. In 3^{rd} *Research Student Workshop, ed. P. Plassmann and P. Roach*, pages 13–19, 2008.

[9] J. A. Azevedo, J. Madeira, M. Costa, E. Q. V. Martins, and F. Pires. A computational improvement for a shortest paths ranking algorithm. *European Journal of Operational Research*, 73:188–191, 1994.

[10] J. Azvedo and E. Q. V. Marins. An algorithm for the multiobjective shortest path problem on acyclic networks. *Investigacao Operational*, 11:52–69, 1991.

[11] A. Bagchi. Route selection with multiple metrics. *Information Processing Letters*, 64:203–205, 1997.

[12] O. Barndorff-Nielsen and M. Sobel. On the distribution of the number of admissible points in a vector random sample. *Theory of Propability and its Applications*, 11(2):283–305, 1966.

[13] C. Barnhart, N. Boland, L. Clarke, E. L. Johnson, G. L. Nemhauser, and Shenoi R. G. Flight string models for aircraft fleeting and routing. *Trans Sci*, 32:208–220, 1998.

[14] R. Batta and S. S. Chiu. Optimal obnoxious path on a network: Transportation of hazardous materials. *Operations Research*, 36:84–92, 1988.

[15] C.F. Bazlamacc and K.S. Hindi. Minimum-weight spanning tree algorithms a survey and empirical study. *Computers & Operations Research*, 28:767–785, 2001.

[16] J. E. Beasley and N. Christofides. An algorithm for the resource constrained shortest path problem. *Networks*, 19:379–394, 1989.

[17] R.E. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958.

[18] J. L. Bentley, H. T. Kung, M. Schkolnick, and C. D. Thompson. On the average number of maxima in a set of vectors and applications. *Journal of ACM*, 25(4):536–543, 1978.

[19] D. P. Bertsekas. A simple and fast label correcting algorithm for shortest path. *Networks*, 23:703–709, 1993.

[20] D. P. Bertsekas. *Network Optimization: Continuous and Discrete Models*. Athena Scientific, 1998.

[21] H. Bettahar and A. Bouabdallah. A new approach for delay-constrained routing. *Computer Communications*, 25(18):1751–1764, 2002.

[22] N. Boland, J. Dethridge, and I. Dumitrescu. Accelerated label setting algorithms for the elementary resource constrained shortest path problem. *Operations Research Letters*, 34:5868, 2006.

[23] R. Borndoefer, M. Grotschel, and A. Lobel. Scheduling duties by adaptative column generation. Technical Report 01-02, Konrad-Zuze-Zentrum fur Informationstechnik, Berlin, 2001.

[24] J. Brumbaugh-Smith and D. Shier. An empirical investigation of some bicriterion shortest path algorithms. *European Journal of Operational Research*, 43(2):216–224, 1989.

[25] J. Buchanan and L. Gardiner. A comparison of two reference point methods in multiple objective mathematical programming. *European Journal of Operational Research*, 149:17–34, 2003.

[26] W. M. Carlyle and R. K. Wood. Lagrangian relaxation and enumeration for solving constrained shortest-path problems. In 38th *Ann ORSNZ Conf*, Hamilton, New Zealand, University of Waikato, November 2003.

[27] R. L. Carraway, T. L. Morin, and H. Moskowitz. Generalized dynamic programming for multicriteria optimization. *European Journal of Operational Research*, 44(1):95–104, 1990.

[28] Z. L. Chen and Powell W. B. A generalized threshold algorithm for the shortest path problem with time windows. In *Network Design: Connectivity and Facilities*, pages 303–318. Pardalos, P. and Du, D., 1998.

[29] B.V. Cherkassky, A.V. Goldberg, and T. Radzik. Shortest paths algorithms: Theory and experimental evaluation. *Mathematical Programming*, 73(2):129–174, 1996.

[30] H. Chin-Chieh, C. Da-Ren, and D. Hua-Yuan. An efficient algorithm for the shortest path problem with forbidden path. *LNCS*, 5574:638–650, 2009.

[31] E.I. Chong, S.R. Maddila, and S.T. Morley. On finding single-source single-destination k shortest paths. In 7^{th} *International Conference Computing and Information*, July 1995.

[32] N. Christofides, A. Mingozzi, and P. Toth. Exact algorithms for the vehicle routing problem based on spanning tree and shortest path relaxations. *Math. Program.*, 20:255–282, 1981.

[33] J. C. N. Climaco, J. M. F. Craveirinha, and M. M. B. Pascoal. A bicriterion approach for routing problem in multimedia network. *Networks*, 41:206–220, 2003.

[34] J. C. N. Climaco, J. M. F. Craveirinha, and M. M. B. Pascoal. An automated reference point-like approach for multicriteria shortest path problem. *Journal of Systems Science and Systems Engineering*, 15(3):314–329, 2006.

[35] J. C. N. Climaco and E. Q. V. Martins. On the determination of the nondominated paths in a multiobjective network problem. *Methods in Operations Research*, 40:255–258, 1981.

[36] J. C. N. Climaco and E. Q. V. Martins. A bicriterion shortest path algorithm. *European Journal of Operational Research*, 11(4):399–404, 1982.

[37] J.F. Cordeau, G. Desaulniers, J. Desrosiers, M.M. Solomon, and F. Soumis. Vrp with time windows. In *The vehicle-routing problem.* Toth, P. and Vigo, D. (Editors), 2002.

[38] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms.* MIT Press/McGraw-Hill Book Company, Boston, MA, second ed. edition, 1992.

[39] J. M. Coutinho-Rodrigues, J. C. N. Climaco, and J. R. Current. A pc-based interactive decision support system fot two objective direct delivery probelms. *Journal of Business Logistics*, 15:305–322, 1994.

[40] J. M. Coutinho-Rodrigues, J. C. N. Climaco, and J. R. Current. An interactive bi-objective shortest path approach: searching for unsupported nondominated solutions. *Computers and Operations Research*, 26(8):789–798, 1999.

[41] J. R. Current and M. Marsh. Multiobjective transportation network design and routing problems: Taxonomy and annotation. *European Journal of Operational Research*, 103:426–438, 1993.

[42] J. R. Current and H. Min. Multiobjective network design of transportation networks: Taxonomy and annotation. *European Journal of Operational Research*, 26:187–201, 1986.

[43] J. R. Current, C. S. Revelle, and J. L. Cohon. An interactive approach to identify the best compromise solution for two objective shortest path problems. *Computers and Operations Research*, 17(2):187–198, 1990.

[44] J.R. Current, H. Pirkul, and E. Rolland. Efficient algorithms for solving the shortest covering path problem. *Transportation Science*, 28:317–327, 1994.

[45] J.R. Current, C.S. Revelle, and J.L. Cohon. The shortest covering path problem: an application of locational constraints to network design. *Journal of Regional Science*, 24:161–185, 1984.

[46] G.B. Dantzig. On the shortest route through a network. *Management Science*, pages 187–190, 1960.

[47] M.S. Daskin. *Network and Discrete Location.* John Wiley and Sons, New York, 1995.

[48] M. Dell'Amico, G. Righini, and M. Salani. A branch-and-price approach to the vehicle-routing problem with simultaneousdistribution and collection. *Transport Sci*, 40(2):235–247, 2006.

[49] N. Deo and C. Pang. Shortest-path algorithms: taxonomy and annotation. *Networks*, 14:275–523, 1984.

[50] M. Desrochers. An algorithm for the shortest path problem with resource constraints. Technical Report G-88-27, GERAD, 1988.

[51] M. Desrochers, J. Desrosiers, and M. Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40(2):342–354, 1992.

[52] M. Desrochers and F. Soumis. A generalized permanent labeling algorithm for the shortest path problem with time windows. *INFOR*, 26:191–212, 1988.

[53] J. Desrosiers, P. Pelletier, and F. Soumis. Plus court chemin avec constraints d'horaires. *RAIRO*, 17:357–377, 1983. in French.

[54] L. Di Puglia Pugliese and F. Guerriero. A class of solution approaches for the linear fractional shortest path problem with time windows. Technical report, University of Calabria, 2008.

[55] R.B. Dial. Algorithm 360: Shortest path forest with topological ordering. *Communications of the ACM*, 12(11):632–633, 1969.

[56] R.B. Dial. A model and algorithm for multicriteria route-mode choice. *Transportation Research Part B: Methodological*, 13(4):311–316, 1979.

[57] E.W. Dijkstra. A note on two problems in connection with graphs. *Numerische Mathematik*, pages 269–271, 1959.

[58] S.E. Dreyfus. An appraisal of some shortest path algorithms. *Operations Research*, 17:395–412, 1969.

[59] M. Dror. Note on the complexity of the shortest path models for column generation in vrptw. *Oper Res*, 42:977–978, 1994.

[60] I. Dumitrescu and N. Boland. Improved preprocessing, labeling and scaling algorithms for the weight-constrained shortest path problem. *Networks*, 42(3):135–153, 2003.

[61] A. Elimam and D. Kohler. Two engineering applications of a constrained shortest path model. *European Journal of Operational Research*, 103:426–438, 1997.

[62] D. Eppstein. Finding the k shortest paths. *SIAM J Compututing*, 28(2):652–673, 1998.

[63] D. Espinoza, R. Garcia, M. Goycoolea, G. L. Nemhauser, and M. W. P. Savelsbergh. Per-seat, on-demand air transportation part i: Problem description and an integer multicommodity flow model. *Transportation Science*, 42(3):263–278, 2008.

[64] D. Feillet, P. Dejax, M. Gendreau, and C. Gueguen. An exact algorithm for the elementary shortest path problem with resource constraints: application to some vehicle routing problems. *Networks*, 43(3):216–229, 2004.

[65] R.W. Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345, 1962.

[66] L.R. Ford and D.R. Fulkerson. *Flows in Networks.* Princeton University Press, Princeton, NJ, 1962.

[67] B. L. Fox. Finding minimum time-cost ratio circuits. *Operations Research*, 17:546–551, 1969.

[68] M.L. Fredman and R.E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, 34(3):596–615, 1987.

[69] H.N. Gabow and R.E. Tarjan. Faster scaling algorithm for network problems. *SIAM J. COMPUT.*, 18(5):1013 – 1036, 1989.

[70] G. Gallo and S. Pallottino. Shortest path algorithms. *Annals of Operations Research*, 13:3–79, 1988.

[71] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman, San Francisco, 1979.

[72] A. Goldberg. Shortest path algorithms: Engineering aspects. In *ISAAC:* 12th *International Symposium on Algorithms and Computation*, pages 502–513, 2001.

[73] A.V. Goldberg. Scaling algorithms for the shortest path problem. In *4th ACM-SIAM Symposium on Discrete Algorithms*, pages 222–231, 1993.

[74] J. Granat and F. Guerriero. The interactive analysis of the multicriteria shortest path problem by the reference point method. *European Journal of Operational Research*, 151(1):103–118, 2003.

[75] F. Guerriero and L. Di Puglia Pugliese. Shortest path problem with forbidden paths: the elementary version. Technical Report 2/09, Laboratorio Logica, 2009.

[76] F. Guerriero and R. Musmanno. Label correcting methods to solve multicriteria shortest path problems. *J. Optim. Theory Appl.*, 111(3):589613, 2001.

[77] F. Guerriero, R. Musmanno, V. Lacagnina, and A. Pecorella. A class of label-correcting methods for the k shortest paths problem. *Operations Research*, 49(3):423–429, 2001.

[78] E. Gutierrez and A. L. Medaglia. Labeling algorithm for the shortest path problem with turn prohibitions with application to large-scale road networks. *Ann Oper Res*, 157(169-182), 2008.

[79] C. Hallam, K. J. Harrison, and J. A. Ward. A multiobjective optimal path algorithm. *Digital Signal Processing*, 11(2):133–143, 2001.

[80] J. Halpern and J. Priess. Shortest paths with time constraints on moving and parking. *Networks*, 4:241–253, 1974.

[81] G. Y. Handler and I. Zang. A dual algorithm for the constrained shortest path problem. *Networks*, 10:293–309, 1980.

[82] P. Hansen. Methods of nonlinear 0-1 programming. *Annals of Discrete Mathematics*, 5:53–70, 1979.

[83] P. Hansen, B. Jaumard, and T. Vovor. Solving the bicriterion shortest path problem from both ends. Technical Report G-98-17, GERAD, 1998.

[84] R. Hassin. Approximation schemes for the restricted shortest path problem. *Math Oper Res*, 17:36–42, 1992.

[85] M. I. Henig. The domination property in multicriteria optimization. *Journal of Mathematical Analysis and Applications*, 114(1):7–16, 1986.

[86] D.J. Houck, J. C. Picard, M. Queyranne, and R. R. Vemuganti. The travelling salesman problem as a constrained shortest path

problem: theory and computational experience. *Oper. Res.*, 17:93–109, 1980.

[87] T. Ibaraki. Enumerative approaches to combinatorial optimization. *Annals of Operations Research*, 11:345–602, 1988.

[88] I. Ioachin, Gélinas S., F. Soumis, and J. Desrosiers. A dynamic programming algorithm for the shortest path problem with time windows and linear node costs. *Networks*, 31(3):193–204, 1998.

[89] S. Irnich. Resource extension functions: properties, inversion and generalization to segments. *OR Spectrum*, 30(1):113–148, 2008.

[90] S. Irnich and G. Desaulniers. Shortest path problems with resource constraints. Technical Report G-2004-11, Les Cahiers du GERAD, 2004.

[91] J. M. Jaffe. Algorithms for finding paths with multiple constraints. *Networks*, 14:95–116, 1984.

[92] B. Jaumard, F. Semet, and T. Vovor. A two-phase resource constrained shortest path algorithm for acyclic graphs. Technical Report G-96-48, Les Cahier du GERAD, 1996.

[93] V. Jimenez and A. Marzal. Computing the k shortest paths: A new algorithm and an experimental comparison. In *Proc* 3^{rd} *Workshop on Algorithm Engineering (WAE99), LNCS 1668*, pages 15–29, 1999.

[94] H. C. Joksch. The shortest route problem with constraints. *J Math Anal Appl*, 14:191–197, 1966.

[95] D. Klingman, A. Napier, , and J. Stutz. Netgen: A program for generating large-scale (un)capacitated assignment, transportation, and minimum cost flow network problems. *Management Science*, 20, 1974.

[96] N. Kohl. *Exact methods for time constrained routing and related scheduling problems*. PhD thesis, Institute of Mathematical Modelling, Technical University of Denmark, DK-2800 Lyngby, 1995. Dissertation no. 16.

[97] E. L. Lawler. *Combinatorial optimization: networks and matroids*. Holt, Rinehart and Winston, New York, 1976.

[98] M. Luquea, K. Miettinen, P. Eskelinen, and F. Ruiz. Incorporating preference information in interactive reference point methods for multiobjective optimization. *Omega*, 37:450 – 462, 2009.

[99] T.L. Magnanti and R.T. Wong. Network design and transportation planning: models and algorithms. *Transportation Science*, 18:1–55, 1984.

[100] E. Q. V. Martins. An algorithm for ranking paths that may contain cycles. *European Journal of Operational Research*, 18:123–130, 1984.

[101] E. Q. V. Martins. An algorithm to determine a path with minimal cost/capacity ratio. *Discrete Appl. Math.*, 8:189–194, 1984.

[102] E. Q. V. Martins. On a special class of bicriterion path problem. *European Journal of Operational Research*, 17:85–94, 1984.

[103] E. Q. V. Martins and J. L. E. Santos. An algorithm for the quickest path problem. *Operations Research Letters*, 20(4):195–198, 1997.

[104] E.Q. Martins, M.M. Pascoal, and J.L. Santos. Deviation algorithms for ranking shortest paths. *International Journal of Foundations of Computer Science*, 10(3):247–261, 1999.

[105] K. Mehlhorn and M. Ziegelmann. Resource constraint shortest paths. In 7th *Ann European Symp on Algorithms (ESA2000), LNCS 1879*, pages 326–337, 2000.

[106] K. Miettinen, M.M. Mäkelä, and K. Kaario. Experiments with classification-based scalarizing functions in interactive multiobjective optimization. *European Journal of Operational Research*, 175:931–947, 2006.

[107] P. B. Mirchandani and M. M. Wiecek. Routing with nonlinear multiattribute cost functions. *Applied Mathematics and Computation*, 54(2-3):215–239, 1993.

[108] P. Modesti and A. Sciomachen. A utility measure for finding multiobjective shortest paths in urban multimodal transportation networks. *European Journal of Operational Research*, 111(3):495–508, 1998.

[109] E.F. Moore. The shortest path through a maze. In *Interantional Symposium on the Theory of Switching*, pages 285–292. Harvard University Press, 1959.

[110] I. Mote, I. Murthy, and D. L. Olson. A parametric approach to solving bicriterion shortest path problems. *European Journal of Operational Research*, 53(1):81–92, 1991.

[111] R. Muhandiramge and N. Boland. Simultaneous solution of lagrangean dual problems interleaved with preprocessing for the weight constrained shortest path problem. *Networks*, 53:358–381, 2009.

[112] I. Murthy and D. L. Olson. An interactive procedure using domination cones for bicriterion shortest path problems. *European Journal of Operational Research*, 72(2):417–431, 1994.

[113] R. Nygaard. *Shortest Path Methods in Representation and Compression of Signals and Image Contours*. PhD thesis, Norwegian University of Science and Technology, 2000.

[114] W. Ogryczak. On goal programming formulations of the reference point method. *Journal of the Operational Research Society*, 52:691–698, 2001.

[115] U. Pape. Implementation and efficiency of moore-algorithm for the shortest path problem. *Math. Programming*, 7:212–222, 1974.

[116] L.L. Pinto and M.M.B. Pascoal. On algorithmsforthetricriteriashortestpathproblemwithtwobottleneck objective functions. *Computers & Operations Research*, 37:1774–1779, 2010.

[117] A. G Qureshi, E. Taniguchi, and T. Yamada. Elementary shortest path problem with resource constraints and time dependent late arrival penalties. *Doboku Gakkai Ronbunshuu D*, 63(4):579–590, 2007.

[118] C. Ribeiro and M. Minoux. A heuristic approach to hard constrained shortest path problems. *Discrete Applied Mathematics*, 10:125–137, 1985.

[119] G. Righini and M. Salani. Symmetry helps: Bounded bidirectional dynamic-programming for the elementary shortest path problem with resource constraints. *Discrete Optimization*, 3(3):255–273, 2006.

[120] G. Righini and M. Salani. New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks*, 51(3):155–170, 2008.

[121] G. Righini and M. Salani. Decremental state space relaxation strategies and initialization heuristics for solving the orienteering problem with time windows with dynamic programming. *Computers & Operations Research*, 36:1191 – 1203, 2009.

[122] J. Roan and C. Lee. Algorithms for linear fractional shortest path problem with time windows. *Pan-Pacific Manage. Rev.*, 6(1):75–84, 2003.

[123] A. Roginsky, K. Christensen, and V. Srinivasan. New methods for shortest path selection for multimedia tra_c with two delay constraints. *Computer Communications*, 22(17):1531–1539, 1999.

[124] C. Romero, M. Tamiz, and D. F. Jones. Goal programming, compromise programming and reference point method formulations: linkages and utility interpretations. *Journal of the Operational Research Society*, 49:986–991, 1998.

[125] L. Santos, J. Coutinho-Rodrigues, and J. R. Current. An improved solution algorithm for the constrained shortest path problem. *Transportation Research Part B*, 41:756–771, 2007.

[126] H. D. Sherali, L. D. Brizendine, T. S. Glickman, and S. Subramanian. Low probability-high consequence considerations in routing hazardous material shipments. *Transp. Sci*, 31:237–251, 1997.

[127] D.R. Shier. On algorithms for finding the k shortest paths in a network. *Networks*, 9(3):195–214, 1979.

[128] A. J. V. Skriver and K. A. Andersen. A label correcting approach for solving bicriterion shortest-path problems. *Computers and Operations Research*, 27(6):507–524, 2000.

[129] H. M. Soroush. The optimal path problem in a bi-attribute network with fractional objective function. *Kuwait J. Sci. Eng.*, 32:35–56, 2005.

[130] H. M. Soroush. Optimal path in bi-attribute networks with fractional objective function. *European Journal of Operational Research*, 190(3):633–658, 2008.

[131] K. Subramani. A zero-space algorithm for negative cost cycle detection in networks. *Journal of Discrete Algorithms*, 5:408–421, 2007.

[132] K. Subramani and L. Kovalchick. A greedy strategy for detecting negative cost cycles in networks. *Future Generation Computer Systems*, 21(4):607–623, 2005.

[133] P. Toth and D. Vigo. Capacitated vehicle-routing problems in the vehicle-routing problem. In *SIAMMonographs on Discrete Mathematics and Applications*. P. Toth and D.Vigo (Editors), 2002.

[134] C. T. Tung and K. L. Chew. A bicriterion pareto-optimal path algorithm. *Asia-Pacific Journal of Operations Research*, 5:166–172, 1988.

[135] C. T. Tung and K. T. Chew. A multicriteria pareto-optimal path algorithm. *European Journal of Operational Research*, 62(2):203–209, 1992.

[136] P. Van Mieghem. Paths in the simple random graph and the waxman graph. *Probability in the Engineering and Information Sciences*, 15:535–555, 2001.

[137] P. Van Mieghem and F. A. Kuipers. On the complexity of QoS routing. *Computer Communications*, 26:376–387, 2003.

[138] D. Villeneuve and G. Desaulniers. The shortest path problem with forbidden paths. *European Journal of Operational Research*, 165(1):97–107, 2005.

[139] Z. Wang. On the complexity of quality of service routing. *Information processing letters*, 69:111–114, 1999.

[140] Z. Wang and J. Crowcroft. Bandwidth-delay based routing algorithms. In *Global Telecommunications Conference*, pages 2129–2133, 1995.

[141] Z. Wang and J. Crowcroft. Quality-of-service routing for supporting multimedia applications. *IEEE Journal on Selected Areas in Communications*, 14(7):1228–1234, 1996.

[142] A. Warburton. Approximation of pareto optima in multiple objective shortest path problems. *Operations Research*, 35:70–79, 1987.

[143] A. Wierzbicki. Basic properties of scalarizing functionals for multiobjective optimization. *Mathematische Operationsforschung und Statistik, s. Optimization*, 8:55–60, 1977.

[144] A. Wierzbicki, M. Makowski, and J. Wessels. *Modelbased Decision Support Methodology with Environmental Applications*. Mathematical Modeling and Applications. Kluwer Academic Publishers, Dordrecht, 2000 edition, 2000.

[145] G. Xue. Primal-dual algorithms for computing weight-constrained shortest paths and weight-constrained minimum spanning trees. In 19th *IEEE International Performance, Computing, and Communications Conference (IPCCC)*, pages 271–277, 2000.

[146] J. Y. Yen. Finding the k-shortest loopless paths in a network. *Manage Sci*, 17:711–715, 1971.

[147] X. Yuan. On the extended bellman-ford algorithm to solve two-constrained quality of service routing problems. In *IEEE International Conference on Computer Communications and Networks*, pages 304–310, Boston, MA, USA, 1999.

[148] X. Yuan. Heuristic algorithms for multiconstrained quality-of-service routing. *IEEE/ACM Transactions on Networking*, 10(2):244–256, 2002.

[149] M. Zabarankin, S. Uryasev, and P. Pardalos. *Optimal risk path algorithms, Cooperative control and optimization*, pages 271–303. Murphey R. and Pardalos P, Kluwer, Dordrecht, 2001.

[150] M. Ziegelmann. *Constrained shortest paths and related problems.* PhD thesis, Universitat des Saarlandes, 2001.