**Les Cahiers du GERAD** ISSN: 0711–2440

# A multidirectional dynamic programming algorithm for the shortest path problem with resource constraints

I. Himmich, I. El Hallaoui,
F. Soumis

G–2018–05

February 2018

La publication de ces rapports de recherche est rendue possible grâce au soutien de HEC Montréal, Polytechnique Montréal, Université McGill, Université du Québec à Montréal, ainsi que du Fonds de recherche du Québec – Nature et technologies.

Dépôt légal – Bibliothèque et Archives nationales du Québec, 2018
– Bibliothèque et Archives Canada, 2018

The publication of these research reports is made possible thanks to the support of HEC Montréal, Polytechnique Montréal, McGill University, Université du Québec à Montréal, as well as the Fonds de recherche du Québec – Nature et technologies.

Legal deposit – Bibliothèque et Archives nationales du Québec, 2018
– Library and Archives Canada, 2018

# A multidirectional dynamic programming algorithm for the shortest path problem with resource constraints

**Ilyas Himmich**

**Issmail El Hallaoui**

**François Soumis**

*GERAD & Département de Mathématiques et Génie Industriel, Polytechnique Montréal (Québec) Canada, H3C 3A7*

ilyas.himmich@gerad.ca
issmail.elhallaoui@gerad.ca
francois.soumis@gerad.ca

**February 2018**
**Les Cahiers du GERAD**
**G–2018–05**

**Abstract:** The shortest path problem with resource constraints finds the least cost path between two nodes in a network while respecting constraints on resource consumption. The problem is mainly used as a subproblem inside column generation for crew scheduling and vehicle routing problems. The standard approach for the subproblems is based on dynamic programming. This class of methods is generally effective in practice when there are only a few resources, but it seems to be time-consuming for huge instances with many resources. To handle this problem, we propose a new exact primal algorithm called the multidirectional dynamic programming algorithm (MDDPA). The proposed approach splits the state space into small disjoint subspaces. These subspaces are sequentially explored in several iterations, where each iteration builds on the previous ones, to reduce the dimension of the subspaces to explore and to quickly generate better paths. Computational experiments on vehicle and crew scheduling instances show the excellent performance of our approach compared to the standard dynamic programming method. In particular, MDDPA is able to generate feasible paths with up to 90% of the optimal cost in less than 10% of the time required by standard dynamic programming. This feature is useful in column generation and may greatly reduce the computational effort, because we can stop the MDDPA solution process once columns with sufficiently negative reduced costs are obtained.

**Keywords:** Transportation, shortest path problem with resource constraints, column generation, directions, dynamic programming

**Acknowledgments:** This work was supported by a Collaborative Research and Development Grant from the Natural Sciences and Engineering Research Council of Canada (NSERC) and Kronos Inc. The authors would like to thank all the members of these organizations for their support and their confidence.

## 1 Introduction

The shortest path problem with resource constraints (SPPRC) aims to find a path between two nodes in a network (a source $s$ and a destination $d$) at minimum cost while respecting restrictions called resource constraints. This problem was introduced by Desrochers et al. [6] as an extension of the classical unconstrained shortest path problem, and it has since attracted the attention of researchers from various domains.

### 1.1 Literature review

Several real-world applications have been modeled as SPPRCs; e.g., military aircraft management [21], railroad management [13], and service routing in communication networks [20]. The problem also appears as a subproblem in column generation (CG) for a huge variety of vehicle routing problems ranging from distribution problems to fleet assignment and crew scheduling such as bus driver scheduling [5] and airline crew pairing [3]. CG is a well-known approach for large vehicle and crew scheduling problems (VCSPs). It is based on a decomposition of the generic formulation of the problem into a master problem (MP), which is generally a set partitioning problem with side constraints, and one or more subproblems. The latter are usually modeled as SPPRCs. They are used to feed potential columns to the MP at each iteration until an optimality criterion is satisfied. These columns are feasible shortest paths with negative reduced costs. They may represent routes, schedules, or planning strategies, depending on the application.

Research into the SPPRC has explored different versions of the problem, with differing numbers of resources and applications. The three versions are: 1. The constrained shortest path problem (CSPP), where a single resource constraint is imposed at the destination node as an upper (or lower) bound on the cumulative consumption of the resource along the chosen path. 2. The shortest path problem with time windows (SPPTW), which also has a single resource but the restrictions are modeled at each node of the network: upper and lower bounds must be respected by every partial path. 3. SPPRC, which is a generalization of SPPTW that considers several resources. For the sake of generality, we consider the SPPRC.

Several solution methods have been proposed. These methods were grouped in [9] into three main classes: k-path ranking methods, Lagrangean relaxation (LR) approaches, and dynamic programming (DP). The k-path ranking methods relax the resource constraints and compute, in increasing order of cost, the shortest paths between two nodes until a feasible path is identified. This approach was introduced in [14] and improved in [19].

LR approaches relax the complex resource constraints before solving a Lagrangean dual problem. The efficiency of these methods depends on the strategies used to reduce the duality gap. Such methods have been developed by Handler & Zang [14], Beasley and Christofides [1], Melhorn and Ziegelmann [16], and Carlyle and Wood [2].

The standard approach for SPPRC is based on DP. The basic algorithm was devised by Desrochers and Soumis [6] as an extension of the Ford–Bellman algorithm with the addition of resource constraints [17]. The algorithm assigns states to each node, where each state at node $i$ refers to a feasible partial path from $s$ to $i$. It then repeatedly extends the states to generate new ones, and it stops when no further extension is possible. Moreover, dominance rules are applied to remove states corresponding to unpromising partial paths. DP-based algorithms for the SPPRC include [6, 7, 8, 4]. These methods are particularly suitable for CG for three reasons. First, unlike LR, DP is able to provide the MP with several columns at a time. Second, it is able to deal with the most complex rules, usually modeled by nonlinear and even nonconvex resource constraints. Third, all the Pareto-optimal paths computed are integer by default. However, the application of DP to large transportation problems has shown that the method suffers from the curse of dimensionality. Many applications, especially in vehicle routing and crew scheduling, involve huge networks with hundreds of thousands of arcs and many resource constraints. DP algorithms create billions of labels during the solution process, and only a few of them are effective. This makes the process extremely time-consuming.

Intense research activity has explored the efficient solution of large problems. A modified version of [8], using preprocessing techniques and Lagrange multipliers, has been proposed [10] for the CSPP. This problem has been addressed by Lozano & Medaglia [15] using a pulse algorithm, this technique enumerates all possible paths and uses pruning strategies to narrow the search space. Nagih and Soumis [17] have investigated the effect of the number of resources on the dominance rules; they compute aggregated resources by projecting the original ones onto a space of smaller dimension. New refinements of the solution of the subproblems using DP have been presented by Feillet et al. [11]. They use a limited-discrepancy search to reduce the size of the state space. They use local rather than global search, and their algorithm extends the labels to a subset of the most promising arcs first, before allowing extension to less promising ones. Another improvement of the DP algorithm, called bidirectional dynamic programming [18], propagates labels both forward from the source to the destination and backward from the destination to the source. The forward and backward labels terminating at the same nodes are then combined to form a complete path.

### 1.2 Motivation and contributions

The motivation for this work is the observation that in CG, the purpose of solving subproblems is to feed the MP with new columns with sufficiently negative reduced costs; they are not necessarily optimal, except in the final iterations. Therefore, we seek a primal method that returns a set of feasible solutions at each iteration while solving the subproblems. This feature is not offered by DP methods unless a heuristic stopping criterion is used, because they explore the entire state space before returning feasible solutions.

The contribution of this paper is threefold. First, we introduce a new formulation of the SPPRC that is suitable for reoptimization using previously generated labels. Second, we propose a new exact algorithm for the SPPRC, called the *multidirectional dynamic programming algorithm* (MDDPA). It is a primal method that returns sets of feasible paths of nonincreasing cost leading to optimality, while performing iterative searches on a reduced subnetwork. Third, we evaluate the performance of our approach compared to standard DP, the most common method in commercial solvers. The tests are performed on VCSP networks with up to 600,000 nodes and 1,000,000 arcs.

MDDPA combines three ideas:

i) A label storing procedure partitions the state space into small disjoint subspaces.
ii) Label loading strategies iteratively explore the subnetworks inducing these subspaces. We evaluate two loading strategies, nearest first and best first.
iii) Learning from locally efficient labels shows how to efficiently use the results of the previous iterations. We use *label fathoming* to prevent the extension of unpromising states and *feasible descent directions* (FDDs) to construct new paths using those previously generated.

The result is an efficient exact method that can quickly return interesting solutions and is able to prove optimality much earlier than standard DP can.

The paper is organized as follows. The next section presents the new generalized mathematical formulation of the SPPRC. Section 3 gives a detailed description of MDDPA. Section 4 reports the results of the computational experiments on simultaneous VCSP instances, and Section 5 provides concluding remarks.

## 2 Generalized mathematical formulation

Consider an acyclic connected network $G(V, A)$ where $V$ is the set of nodes including the source and destination nodes $s$ and $d$, and $A$ is the set of arcs. Let $R$ be the set of resources. For each arc $(i, j) \in A$, in addition to its cost $c_{ij}$, there is an $|R|$-dimensional resource consumption vector $(r_{ij}^1, r_{ij}^2, ..., r_{ij}^{|R|})$. We denote by $R_i^t$ the consumption of each resource $t \in R$ over all the arcs composing a partial path $\pi_i$ from $s$ to $i$, while $a_i^t$ and $b_i^t$ are the lower and upper bounds on the resource $t \in R$ at node $i$. The SPPRC finds a least cost path among all the paths from $s$ to $d$ that satisfy the resource constraints induced by $R$.

## 2.1 Standard formulation

The SPPRC is formulated as follows:

$$(P_1) \quad \text{Minimize} \quad \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{1}$$

s.t.

$$\sum_{i\in V} x_{ij} - \sum_{i\in V} x_{ji} = \begin{cases} -1 & \text{if } j = s \\ 0 & \text{if } j \in V \setminus \{s,d\} \\ 1 & \text{if } j = d \end{cases} \tag{2}$$

$$x_{ij}(R_i^t + r_{ij}^t - R_j^t) \leq 0 \quad \forall t \in R, \forall (i,j) \in A \tag{3}$$

$$a_i^t \leq R_i^t \leq b_i^t \quad \forall t \in R, \forall i \in V \tag{4}$$

$$x_{ij} \in \{0,1\} \quad \forall (i,j) \in A \tag{5}$$

Constraints (2) are the flow conservation constraints. Constraints (3) model the resource consumption along arc $(i,j)$ whenever it is part of the solution (path), while constraints (4) require the resource consumption along the $s$-$i$ partial path to be within the corresponding resource interval. Note that it is permissible to arrive at a node $i \in V$ even if $R_i^t < a_i^t$ for some $t \in R$; in this case $R_i^t$ takes the value $a_i^t$. Constraints (5) are the binary requirements on the arc flow variables $x_{ij}$, $(i,j) \in A$.

In what follows, we reformulate the flow conservation constraints as set partitioning constraints. The purpose is twofold: first, the reformulation offers a new measure of distance between each node in the network and the destination node, and shows the length of each arc using this measure; second, it allows us to introduce a new generalized formulation of the SPPRC. This formulation makes possible the partition of the state space that we propose, while the measure of distance is useful for exploring the elements of this partition.

## 2.2 Mathematical reformulation of the flow conservation constraints

We now reformulate constraints (2.2). We sort the nodes of $G(V,A)$ in topological order and refer to each node by its rank in this order. Thus, $V = \{1,2,...,|V|\}$ is the set of ordered nodes, where the source and destination nodes are indexed respectively by 1 and $|V|$. We define the notion of a *cocycle* as follows.

**Definition 1** *Consider a node $k \in V$. Let the $k^{th}$ cocycle, denoted $Co_k$, be the set of all arcs $(i,j) \in A$ such that the origin $i$ is topologically ordered before node $k$, and the destination $j$ is ordered strictly after node $k$. Formally, $Co_k = \{(i,j) \in A | i \leq k < j\}$.*

With each cocycle, we associate the following cut called a cocycle constraint:

$$\sum_{(i,j)\in Co_k} x_{ij} = 1 \qquad \forall k \in \{1,2,...,|V|-1\}. \tag{6}$$

Figure 1 shows the cocycle constraints on a four-node acyclic network. The next proposition shows that the cocycle constraints are sufficient to ensure connectivity of the path, i.e., flow conservation (of one unit) on the path.

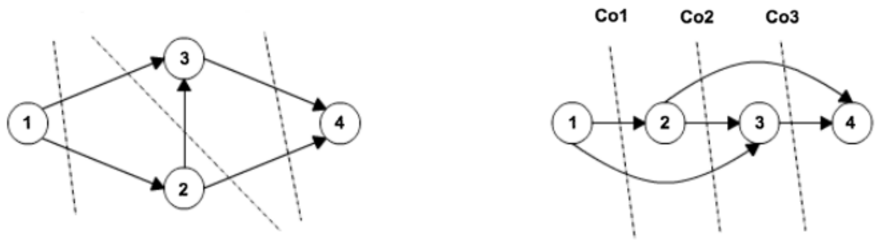

**Figure 1: Cocycle constraints.**

**Proposition 1** *The cocycle constraints are equivalent to the flow conservation constraints in acyclic connected networks.*

**Proof.** Let $A^+(i)$ and $A^-(i)$ be the sets of arcs leaving and entering node $i$, respectively. For simplicity, we use $x_a$ instead of $x_{ij}$ to refer to the variable for arc $(i,j)$. We have $Co_k = \cup_{i\leq k} A^+(i) \setminus \cup_{i\leq k} A^-(i), \forall k \in \{1, 2, ..., |V|-1\}$. Thus, $\sum_{a\in Co_k} x_a = \sum_{i\leq k}\sum_{a\in A^+(i)} x_a - \sum_{i\leq k}\sum_{a\in A^-(i)} x_a, \forall k \in \{1, 2, ..., |V|-1\}$. For the nodes indexed by 1 and —V—, we have $Co_1 = A^+(1)$ and $Co_{|V|-1} = A^-(|V|)$. Thus, $\sum_{a\in Co_1} x_a = 1 \iff \sum_{a\in A^+(1)} x_a = 1$, and $\sum_{a\in Co_{|V|-1}} x_a = 1 \iff \sum_{a\in A^-(|V|)} x_a = 1$.

Let us prove the equivalence for any node $k \in \{2, ..., |V|-1\}$.

$\Rightarrow$ Suppose that $x$ is a solution that satisfies the cocycle constraints but not the flow conservation constraints. Then $\exists\ k$ such that $\sum_{a\in A^+(k)} x_a - \sum_{a\in A^-(k)} x_a = Q \neq 0$. We have $\sum_{i\leq k}(\sum_{a\in A^+(i)} x_a - \sum_{a\in A^-(i)} x_a) = \sum_{i\leq k-1}(\sum_{a\in A^+(i)} x_a - \sum_{a\in A^-(i)} x_a) + \sum_{a\in A^+(k)} x_a - \sum_{a\in A^-(k)} x_a$. Since $\sum_{a\in Co_k} x_a = \sum_{i\leq k}\sum_{a\in A^+(i)} x_a - \sum_{i\leq k}\sum_{a\in A^-(i)} x_a = 1$, $\forall k \in \{1, 2, ..., |V|-1\}$, we must have $1 = 1 + Q$, which is false unless $Q = 0$.

$\Leftarrow$ $\sum_{a\in Co_k} x_a = \sum_{i\leq k}\sum_{a\in A^+(i)} x_a - \sum_{i\leq k}\sum_{a\in A^-(i)} x_a$
$= \sum_{i\leq k}(\sum_{a\in A^+(i)} x_a - \sum_{a\in A^-(i)} x_a)$
$= \sum_{a\in A^+(1)} x_a - \sum_{a\in A^-(1)} x_a + \sum_{1<i\leq k}(\sum_{a\in A^+(i)} x_a - \sum_{a\in A^-(i)} x_a)$
$= 1$.

Thus, the two formulations are equivalent. □

Hence, the shortest path problem may be seen as a set partitioning problem with side constraints. Each column represents an arc, while the cocycle equality constraints ensure that every path covers each cocycle exactly once. Problem $(P_1)$ becomes:

$$\begin{array}{lll} (P_2) & \text{Minimize} & \sum_{(i,j)\in A} c_{ij}x_{ij} \\ & \text{s.t.} & \sum_{(i,j)\in Co_k} x_{ij} = 1 \quad \forall k \in \{1, 2, ..., |V|-1\} \\ & & (2.3)\text{–}(2.5) \end{array}$$

## 2.3 Generalized mathematical formulation

The standard approach to the SPPRC is based on DP, as mentioned above. A DP algorithm associates with each partial path $\pi_{si}$ from $s$ to node $i$ a state indicating the cumulative cost and resource consumption. Each state is represented by an $|R+1|$-vector $l = [C_l, R_l^1, R_l^2, ..., R_l^{|R|}]$ called a label. New labels are dynamically created by extending the existing ones. The labels are updated after each extension according to a resource extension function, $f_{ij} : \mathbb{R}^{|R+1|} \to \mathbb{R}^{|R+1|}$. The classical extension function is defined as follows: $f_{ij}(l) = l + [c_{ij}, r_{ij}^1, r_{ij}^2, ..., r_{ij}^{|R|}] = l'$.

An extension is valid only if the new label $l'$ is feasible in terms of the resource constraints; otherwise, it is eliminated. Feasible labels may be suppressed by the dominance rules. Dominance rules are used to compare each pair of feasible partial paths arriving at a given node; the unpromising one is discarded.

**Definition 2** *Let $l_1$ and $l_2$ be two feasible labels associated with two partial paths from $s$ to node $i$. We say that $l_2$ is dominated by $l_1$ if and only if $C_1 \leq C_2$ and $R_1^t \leq R_2^t$ $\forall t \in R$ and at least one inequality is strict.*

**Definition 3** *A label $l$ at node $i$ is efficient if it is feasible and not dominated by any other label at node $i$. A partial path is said to be efficient if it corresponds to an efficient label.*

We associate with each node $i$ in the network a set $\mathcal{S}_i$ of efficient labels. We denote by $i^*$ the index of the first node $i$ in the topological order such that $\mathcal{S}_i \neq \emptyset$. The resulting problem is a generalization of the classical SPPRC, which has one label $l_1 = [0, 0, ..., 0]$ at the source node, and no labels at the remaining nodes, i.e., $\mathcal{S}_1 = \{l_1\}$ and $\mathcal{S}_i = \emptyset$, $\forall i \neq 1$. The model is as follows:

$$(P_3)\ \text{Minimize} \quad \sum_{(i,j)\in A, i\geq i^*} c_{ij}x_{ij} + \sum_{i\in V, i\geq i^*, l\in\mathcal{S}_i} c_i^l y_i^l \tag{7}$$

$$\text{s.t.} \sum_{i\geq i^*, l\in\mathcal{S}_i} y_i^l = 1 \tag{8}$$

$$\sum_{(i,j)\in Co_k} x_{ij} + \sum_{i>k, l\in\mathcal{S}_i} y_i^l = 1 \qquad \forall k \in \{i^*, i^*+1, ..., |V|-1\} \tag{9}$$

$$y_i^l(r_l^t - R_i^t) \leq 0 \qquad \forall i \in V, i \geq i^*, \forall l \in \mathcal{S}_i, \forall t \in R \tag{10}$$

$$x_{ij}(R_i^t + r_{ij}^t - R_j^t) \leq 0 \qquad \forall t \in R, \forall (i,j) \in A, i \geq i^* \tag{11}$$

$$a_i^k \leq R_i^t \leq b_i^t \qquad \forall t \in R, \forall i \in V, i \geq i^* \tag{12}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A, i \geq i^* \tag{13}$$

$$y_i^l \in \{0,1\} \qquad \forall i \in V, i \geq i^*, \forall l \in \mathcal{S}_i \tag{14}$$

Here $y_i^l$ is the decision variable that indicates whether or not the label $l$ terminating at node $i$ is used to construct an optimal path, $c_i^l$ is its cumulated cost, and $r_l^t$ is its cumulated consumption of resource $t$. We recall that each label in $\mathcal{S}_i$ represents a feasible partial path from $s$ to $i$ that obviously covers all the cocycles $Co_k$ for $k \in \{1, 2, ..., (i-1)\}$.

Constraint (8) ensures that exactly one stored label is selected from $\cup_{i\in V}\mathcal{S}_i$, so that its corresponding partial path is part of an optimal solution. In particular, if $i^* = s$, the initial label $l_1 = [0, 0, ..., 0] \in \mathcal{S}_1$ may be the selected label. Constraints (9) ensure that all the cocycles $Co_k, k \geq i^*$ are covered exactly once. Each cocycle may be covered either by labels $l \in S_i, i \geq i^*$ or using an arc $(i,j) \in Co_k$. The cocycles $Co_k, k < i^*$ are obviously covered by the stored labels $l \in S_i, i \geq i^*$. Constraints (10) associate with $R_i^t$ the cumulated consumption of resource $t \in R$, if a label $l \in \mathcal{S}_i$ is chosen to be part of an optimal solution. Constraints (11) and (12) are subsets of constraints (3) and (4) with the restriction $i \geq i^*$. Constraints (13) and (14) are integrality constraints.

We observe that this formulation may be used to model different real-world situations. For example, re-optimization is often necessary when the network is affected by minor changes such as updates to the cost or resource consumptions of a subset of arcs, or the removal or addition of arcs or nodes. In particular, in the CG context, a subset of arcs will have reduced cost changes because of modifications to the dual values. In these cases, we have in hand a set of labels generated at previous iterations or during a previous solution process. Some of these labels are not affected by the minor perturbations and can be reused with little computational effort to produce new solutions. The affected part of the labels may be used in a heuristic framework after their attributes have been updated if possible.

Given that the initial labels in $\mathcal{S}_i$ for $i \in V$ correspond to efficient partial paths from the source node to a given node $i$, a first observation is that an efficient feasible path from $s$ to $d$ can be constructed using a completion of any previously existing label at any node in the network, not only the initial label $l_1 \in \mathcal{S}_1$.

Also, the extension of each label can be done independently of the rest of the labels. The formulation $(P_2)$ then offers a disjoint partition of the solution space. These two observations are fundamental to MDDPA.

## 3 Solution approach

### 3.1 Motivation

Starting with an initial label $l_1 = [0, 0, ..., 0]$ at the source node $s$, and empty sets of labels at all other nodes, DP algorithms seek efficient labels by extending the partial paths for the existing efficient labels at a given node toward the outgoing arcs. The algorithm stops when all the efficient labels reach the destination node. These labels are said to be optimal in the Pareto sense.

MDDPA can handle the problem of dimensionality without increasing the complexity in the worst case. The idea is to split the state space into several subspaces and to solve them iteratively using the generalized mathematical formulation (Section 2.3). The core of our approach is an initialization step in which we store sets of labels at the nodes of the network, before extending these labels iteratively using DP. MDDPA also allows the use of results from previous iterations to tighten the dimension of the search subspace in the current iteration and to construct new solutions using previous ones.

### 3.2 MDDPA algorithm

Standard DP is a monodirectional search, in the sense that the labels are extended starting from the initial label $l_1 = [0, 0, ..., 0] \in \mathcal{S}_1$. Multidirectional search provides a framework for using DP in small subspaces and for performing searches in several directions. In an initialization step it provides each node in the network with a set of efficient labels $\mathcal{S}_i$ (which may be empty). These labels are then iteratively loaded and extended from their resident nodes $i \in V$ to the destination node $d$, following predetermined loading rules. Some of the previously generated labels are used to fathom unpromising labels in the current iteration, while others define FDDs that are useful for constructing new feasible paths. Therefore, MDDPA is based on three ideas: a label storing procedure, label loading strategies, and the ability to learn from locally efficient labels.

We need the following notation: $k$ is the iteration number, and for each node $i \in V$, $\mathcal{S}_i$ is the set of stored labels ($\mathcal{S} = \bigcup_i \mathcal{S}_i$), $\mathcal{L}_i$ is the set of active labels, i.e., labels to extend ($\mathcal{L} = \bigcup_i \mathcal{L}_i$), and $\mathcal{P}_i$ is the set of locally efficient labels (Definition 4). In addition, $\Pi_i$ is the subset of $\mathcal{P}_i$ that contains all the locally efficient labels that have contributed to the construction of a feasible path in a previous iteration. We denote by $\delta_i^+$ the set of outgoing arcs from node $i$. The procedure $LSP(\bar{G}, \mathcal{S})$ is the label storing procedure defined in Section 3.2.1. $LLS(\mathcal{L},\mathcal{S},i^*,k)$ is the procedure that loads the selected labels to extend from $\mathcal{S}$ at a given iteration $k$ and returns the index $i^*$ of the first node in the topological order with $\mathcal{L}_i \neq \emptyset$; it is defined in Section 3.2.2. $Dominance(\mathcal{L}_i)$ is the dominance function (Definition 2); it applies the dominance rules at a given node $i$ to fathom the unpromising labels and retain the efficient ones from the set of labels $\mathcal{L}_i$. The function $Extension(\mathcal{L}_i, j)$ extends the labels in $\mathcal{L}_i$ and returns the newly created labels at node $j$ after checking their feasibility in terms of the cost bound and resource constraints (Section 2.3). $LLEL(\mathcal{L}_i, \mathcal{P}_i, C^{best})$ is a procedure that fathoms unpromising labels and identifies FDDs using locally efficient labels in $\mathcal{P}_i$; this is explained in Section 3.2.3.

Finally, $Cost\_Bounding(\Pi_d, C^{best})$ is the dynamic cost-bounding procedure; it tightens the upper bounds at the nodes using the cost of the best path from $\Pi_d$. A label at node $i$ can be extended to successor nodes only if its cost is at most the upper bound at node $i$. These upper bounds, denoted $\bar{C}_i$, are dynamically updated whenever a feasible path $\pi$ with a better cost is identified. They are computed at each node $i \in V$ as follows: $\bar{C}_i = C^{best} - C_i^{sp}$, where $C^{best} = min\{C_\pi, \pi \in \Pi_d\}$ is the cost of the best feasible path, and $C_i^{sp}$ is the cost of the shortest path from the current node $i$ to the destination node. The shortest paths $C_i^{sp}, i \in V$ are easily computed in a preprocessing step, with a backward call of the Ford–Bellman algorithm from the destination node.

Algorithm 1 presents the MDDPA pseudocode. We note that the procedure $Cost\_Bounding(C^{best})$ is called at two levels of the MDDPA: at the end of each iteration, and inside the procedure $LLEL(\mathcal{L}_i, \mathcal{P}_i, C^{best})$

if the latter found a path with a better cost than the existing best one. In addition to its ability to discard unpromising labels, the dynamic cost bounding ensures the generation of a sequence of feasible paths of nonincreasing cost leading to optimality.

**Algorithm 1:** MDDPA

```
//Initialization //
Define a subnetwork $\bar{G}(\bar{V}, \bar{A})$ of $G(V, A)$
Compute the reverse shortest path from $d$ to $s$
$C^{best} \leftarrow \infty$
for all $i \in V \setminus \{1\}$ do
    $\mathcal{L}_i \leftarrow \emptyset$ , $\mathcal{S}_i \leftarrow \emptyset$ , $\mathcal{P}_i \leftarrow \emptyset$
$\mathcal{S} \leftarrow LSP(\bar{G}, \mathcal{S})$
$k \leftarrow 1$ , $i^* \leftarrow |V| - 1$
//Search procedure //
repeat
    $LLS(\mathcal{L}, \mathcal{S}, i^*, k)$
    for $i = i^*$ to $d$ do
        $Dominance(\mathcal{L}_i)$
        $LLEL(\mathcal{L}_i, \mathcal{P}_i, C^{best})$
        for all $(i, j) \in \delta_i^+$ do
            $\mathcal{L}_j \leftarrow \mathcal{L}_j \cup Extension(\mathcal{L}_i, j)$
        $\mathcal{P}_i \leftarrow \mathcal{P}_i \cup \mathcal{L}_i$
    $Cost\_Bounding(C^{best})$
    $k \leftarrow k + 1$
until $\mathcal{S} = \emptyset$
```

### 3.2.1 Label storing procedure (LSP)

The main purpose of LSP is to allow the SPPRC to be formulated as $(P_3)$. We need to feed the sets of stored labels $\mathcal{S}_i, i \in V$ with labels representing feasible partial paths from $s$ to $i$. Let $\bar{G}(\bar{V}, \bar{A})$ be a subnetwork of $G(V, A)$ such that $\bar{V} \subset V$ and $\bar{A} \subset A$. A node $j \in V$ is said to be a neighbor of $\bar{G}$ if there is an arc $(i, j) \in A \setminus \bar{A}$ such that $i \in \bar{V}$. Algorithm 2 is used to fill the sets of stored labels associated with neighbors of $\bar{G}$.

**Algorithm 2:** Label Storing Procedure $LSP(\bar{G}, \mathcal{S})$

```
Initialization. $\mathcal{S}_i \leftarrow \emptyset, \forall i \in V$; $\mathcal{L}_1 \leftarrow \{[0, 0, ..., 0]\}$; $\mathcal{L}_i \leftarrow \emptyset \ \forall i \in V \setminus \{1\}$
for all $i \in \bar{V}$ do
    $Dominance(\mathcal{L}_i)$
    for all $(i, j) \in \delta_i^+$ do
        $\mathcal{T}_j \leftarrow Extension(\mathcal{L}_i, j)$
        if $(i, j) \in \bar{A}$ then
            $\mathcal{L}_j \leftarrow \mathcal{L}_j \cup \mathcal{T}_j$
        else
            $\mathcal{S}_j \leftarrow \mathcal{S}_j \cup \mathcal{T}_j$
return the set of labels in $\mathcal{S}_i \ \forall i \in V$
```

Starting from the initial label $l_1 = [0, 0, ..., 0] \in \mathcal{L}_1$ at the source node, Algorithm 2 is a modified version of standard DP. It calls the dominance function at a node $i \in \bar{V}$, extends the efficient labels to the adjacent nodes $j \in V$ using arcs $(i, j) \in \delta_i^+$, and saves the newly created labels in a temporary list $\mathcal{T}_j$. If arc $(i, j) \in \bar{A}$, $\mathcal{T}_j$ is added to $\mathcal{L}_j$, to be extended later. Otherwise, node $j$ is a neighbor of $\bar{G}$, so $\mathcal{T}_j$ is immediately stored in $\mathcal{S}_j$. By the end of Algorithm 2, all the neighbors have sets of labels $\mathcal{S}_i$. This structure allows the SPPRC to be formulated as a $(P_3)$ problem.

**Remark 1** *Let $\mathcal{S}_i$, $i \in V$ be the set of labels generated by Algorithm 2. The sets of paths generated by the extension of labels in $\mathcal{S}_i$ $\forall i \in V$ are disjoint, in the sense that we cannot obtain the same path by extending two different labels in $\cup_{i \in V} \mathcal{S}_i$.*

We observe that $\bar{G}(\bar{V}, \bar{A})$ can be chosen to be any connected subgraph of $G$ containing the source and destination nodes. In a CG context, we construct $\bar{G}$ using the paths for the nondegenerate basic columns (variables) of the MP. First, these paths cover in a balanced way all the regions of the network, since they form a feasible solution to the MP. Second, they have zero reduced costs, so we consider that their sequences of arcs and nodes are better able to contribute to the generation of new paths with negative reduced costs.

**Proposition 2** *Using the sets of labels constructed by Algorithm 2, the formulation $(P_1)$ is equivalent to $(P_3)$, in the sense that every feasible solution for one is also feasible for the other. In particular, the two formulations have the same optimal solution.*

**Proof.** Let $\boldsymbol{x^\pi}$ be a feasible solution to $(P_1)$, and $\pi$ the corresponding path, there is a sequence of feasible labels in $G$: $\{l_i,\ i \in \mathcal{N}^\pi\}$, where $\mathcal{N}^\pi$ is the set of nodes traversed by $\pi$. Let $v \in \mathcal{N}^\pi$ be the first node of path $\pi$ (in topological order) that is a neighbor of $\bar{G}$. The corresponding label $l_v = [c^l, r_l^1, r_l^2, ..., r_l^{|R|}]$ is then feasible in $G$ and therefore feasible in $\bar{G}$, so it is stored in $\mathcal{S}_v$ by Algorithm 2. If we set $y_v^l = 1$ and $y_i^{l'} = 0, \forall l' \neq l, \forall i \in V$, and we set $R_v^t$ to $r_l^t$ for each $t \in R$, constraints (2.8) and (2.10) are verified, and constraints (2.9) are verified for $k \leq v$. Moreover, $(P_3)$ becomes a restricted problem of $(P_1)$ with $v$ as source node and $l_v$ as initial label. So, $\boldsymbol{x^\pi}$ verifies obviously the rest of constraints.

Suppose now that $(\boldsymbol{x^\pi}, \boldsymbol{y^\pi})$ is a feasible solution to $(P_3)$ and $\pi$ the corresponding path. Let $\mathcal{A}^\pi$ be the set of arcs composing $\pi$. First, if we set $x_{ij} = 1\ \forall (i,j) \in \mathcal{A}^\pi$ and $x_{ij} = 0\ \forall (i,j) \notin \mathcal{A}^\pi$, constraints (2.2) are clearly verified. Second, there is necessarily a node $v \in V$ and a label $l_v \in \mathcal{S}_v$ such that $y_v^l = 1$ and $y_i^{l'} = 0, \forall l' \neq l, \forall i \in V$. Label $l_v$ is feasible in $\bar{G}$, it is then feasible in $G$, so constraints (2.3) and (2.4) are verified for each $i \leq v$, and constraints (2.3) and (2.4) are verified for $i > v$ are the same as in $(P_3)$. This completes the proof. □

On the one hand, Proposition 2 proves that solving $(P_1)$ is equivalent to solving $(P_3)$. On the other hand, Remark 1 shows that the sets of stored labels $(\mathcal{S}_i)_{i=1,...,|V|-1}$ offer a disjoint partition of the solution space of the SPPRC. These two results form the core of MDDPA, which iteratively solves $(P_3)$, using the label loading strategies discussed below.

#### 3.2.2 Label loading strategies (LLS)

With each set of labels $\mathcal{S}_i$ generated by Algorithm 2 at node $i$ we associate a restricted search space induced by the subnetwork $G_i(V_i, A_i)$ of $G(V, A)$, where $V_i = \{j \in V, j \geq i\}$ and $A_i = \{(j,k) \in A, j \geq i\}$. These search spaces can be explored with various label loading strategies, and the order in which the sets of labels $\mathcal{S}_i, i \in V$ are extended is important since it may have a considerable impact on the effectiveness of the overall algorithm. We propose two loading strategies: *Nearest First* is based on the distance from the destination node, and *Best First* is based on the labels' costs.

##### 3.2.2.1 Nearest first (NF) strategy

The NF strategy extends the labels $\mathcal{S}_i, i \in V$ while prioritizing labels that are closer to the destination node. The distance from the destination node is computed using the number of uncovered cocycles. Recall that labels in $\mathcal{S}_i$ correspond to partial paths that cover the cocycles $Co_1, Co_2, ..., Co_{i-1}$. When we add complementary sequences of arcs that cover the remaining cocycles $Co_i, Co_{i+1}, ..., Co_{|V|-1}$, we form complete paths from $s$ to $d$.

Clearly, the labels stored at the nodes topologically ordered at the end of the network correspond to partial paths covering the largest portions of cocycles. They therefore need shorter sequences of arcs to form complete paths. Furthermore, extending these labels before those that are relatively far from the destination node could generate paths quickly, because the search spaces induced by the subnetworks $G_i(V_i, A_i)$ are of limited dimensions and so require limited computational effort.

The NF loading rule extends the labels in $\mathcal{S}_i$ node by node (or set by set) in reverse topological order of the nodes. Formally, the sets of labels in $\mathcal{S}_j$ are extended before those in $\mathcal{S}_i$ if $i < j$ (node $i$ is topologically ordered before node $j$). Since there is no guarantee that the extension of $\mathcal{S}_i$ will lead to feasible paths, and since the extension of each set of labels individually may be inefficient, MDDPA extends several sets of labels at each iteration. Algorithm 3 gives the NF strategy.

**Algorithm 3:** $LLS(\mathcal{L},\mathcal{S},i^*,k)$ using NF strategy

```
$p_k \leftarrow p_0.r^k$
$i \leftarrow i^* - 1$
**repeat**
    $\mathcal{L}_i \leftarrow \mathcal{S}_i$
    $\mathcal{S}_i \leftarrow \emptyset$
    $\mathcal{L} \leftarrow \mathcal{L} \cup \mathcal{L}_i$
    $i \leftarrow i - 1$
**until** $|\mathcal{L}| \geq p_k$
$i^* \leftarrow i$
```

Algorithm 3 uses a sequence of jumps ($p_k$) to extend the selection of labels at each iteration $k$. The lengths of the jumps are defined using the geometric sequence $p_k = p_0.r^k$ with a first term $p_0$ and a common ratio $r > 1$. Formally, if $i^*$ is the first node in the topological order from which the labels were extended at iteration $k-1$, and $p_k$ is the jump to perform at iteration $k$, the labels to load at the $k^{th}$ iteration are $\mathcal{S}_{i^*-1}$, $\mathcal{S}_{i^*-2}$, $\mathcal{S}_{i^*-3}$,...,$\mathcal{S}_j$, such that $\sum_{i=j}^{i=i^*-1} |\mathcal{S}_i| \geq p_k$ and $\sum_{i=j+1}^{i=i^*-1} |\mathcal{S}_i| < p_k$. The use of this sequence makes the last jumps larger, which implies the exploration of large subspaces during the final iterations. This makes the algorithm more efficient for two reasons. First, as a primal method, MDDPA becomes rich in primal information during the final iterations. This information, extracted from the rapid early iterations, is used to tighten the cost bounds and to fathom large portions of unpromising labels, which can substantially reduce the dimensions of the subspaces in the final iterations and consequently reduce the computational complexity of these iterations. Second, we expect MDDPA to return an optimal solution before the end of the process. Therefore, it is always better to do large searches in the final iterations, since the purpose is mainly to prove optimality.

#### 3.2.2.2 Best first (BF) strategy

In the BF strategy, the labels to extend at each iteration are loaded according to their cumulated costs instead of their distance from the destination node. This allows the most promising labels, i.e., those with the lowest costs, to be extended first. These labels are expected to generate paths with interesting reduced costs. This is an appropriate strategy for CG, where columns with good reduced costs are sufficient, especially in the early iterations.

To determine which labels to extend at iteration $k$ of MDDPA, we first sort the set of labels $\mathcal{S} = \cup_{i \in V} \mathcal{S}_i$ in increasing order of cumulated cost. Only the most promising labels are extended at each iteration. Furthermore, the lengths of the jumps to perform on $\mathcal{S}$ at iteration $k$ are dynamically updated according to a geometric sequence $p_k = p_0.r^k$, where $p_0$ is the first jump and $r > 1$ is a common ratio. The stored labels that were extended from $\mathcal{S}$ in previous iterations are deleted so that they cannot be reused in the current iteration. Note that the jumps become longer between iterations, but the sizes of the subspaces to explore in the final iterations do not have the same rate of increase. This is because considerable portions of the labels are expected to be deleted by the label fathoming techniques (Section 3.2.3), which reduces the computational complexity. Algorithm 4 gives the BF loading strategy.

**Algorithm 4:** $LLS(\mathcal{L},\mathcal{S},i^*,k)$ using BF strategy

```
Sort $\mathcal{S}$ in increasing order of cost
$p_k \leftarrow p_0.r^k$
let $l^* = [C_{l^*}, R^1_{l^*}, R^2_{l^*}, ..., R^{|R|}_{l^*}]$ be the label of rank $p_k$ in $\mathcal{S}$
**for all** $i \in V$ **do**
    **for all** $l \in \mathcal{S}_i$ **do**
        **if** $C_l \leq C_{l^*}$ **then**
            $\mathcal{L}_i \leftarrow \mathcal{L}_i \cup \{l\}$
            $\mathcal{S}_i \leftarrow \mathcal{S}_i \setminus \{l\}$
$i^* = min\{i, \mathcal{L}_i \neq \emptyset\}$
```

### 3.2.3 Learning from locally efficient labels (LLEL)

Learning refers to techniques that use available information to improve the solution process. This information can either be extracted from knowledge of the problem or generated at previous iterations. The efficiency of learning techniques depends on the quality of this information and on the ability of the algorithm to improve this quality as the iterations proceed. MDDPA, a primal method, is able to dynamically update the primal information and improve its quality, whereas standard DP is not able to use such information.

In this section, we introduce techniques that use the previously generated labels to fathom unpromising labels and to construct feasible improving paths. We first give the following definition.

**Definition 4** *A label $l_i$ at node $i$ is said to be locally efficient if it is efficient at a given iteration of MDDPA. The corresponding partial path is called a locally efficient partial path. The set of locally efficient labels at node $i$ is denoted $\mathcal{P}_i$.*

#### 3.2.3.1 Label fathoming using locally efficient labels

In practice, when we solve SPPRC using DP, the number of labels grows exponentially with the size of the network and the number of resources. Most labels correspond to partial paths that will not prove useful. We use the term label fathoming to refer to tools that recognize and stop the extension of unpromising labels.

Locally efficient labels are labels that have passed the dominance tests in a previous iteration and could potentially dominate new labels. The fathoming technique uses these labels to dominate new ones and therefore reduces the search space in the subsequent iterations. Proposition 3 justifies the validity of dominance using these labels and indicates their usefulness.

**Proposition 3** *If label $l^{'} \in \mathcal{P}_i$ dominates $l \in \mathcal{L}_i$, then $l$ can safely be discarded without affecting optimality.*

**Proof.** The proof is simple. If $l^{'}$ dominates $l$, then no extension of $l$ will lead to a path better than that obtained while extending $l^{'}$ using the same extension function. If $l^{'}$ is dominated by another label $l^{''}$, then $l^{''}$ dominates $l$ as well. Therefore, discarding $l$ does not affect optimality. $\square$

Proposition 4 shows that locally efficient labels are able to discard labels that were not eliminated by dynamic cost bounding.

**Proposition 4** *Locally efficient labels are able to dominate new labels that were not eliminated by cost bounding.*

**Proof.** Suppose that $C^{best}$ is the cost of the best feasible path found in a given iteration. Consider a label $l = [C_l, R_l^1, R_l^2, ..., R_l^{|R|}]$ at node $i$, and let $C_i^{sfp}$ and $C_i^{sp}$ be respectively the cost of the shortest feasible partial path and the cost of the shortest unconstrained partial path from $i$ to $d$. By definition, $C_i^{sfp} \geq C_i^{sp}$, so $C^{best} - C_i^{sfp} \leq C^{best} - C_i^{sp}$. If $C^{best} - C_i^{sfp} \leq C_l \leq C^{best} - C_i^{sp}$, then label $l$ is unpromising and cannot be eliminated by cost bounding at node $i$. However, it can be dominated if there is a locally efficient label dominating it, as shown in Proposition 3. $\square$

Recall that MDDPA also uses cost bounding to discard unpromising labels. Dominance using locally efficient labels has three main advantages. First, it involves the resource consumption of labels, while cost bounding is based only on the cost criterion. Second, the locally efficient labels have already resisted dominance and shown good potential during the previous iterations. They are efficient, and some of them have contributed to the construction of feasible paths. These effective sequences of arcs are therefore more reliable and better able to dominate other labels terminating at their resident nodes in subsequent iterations. Third, locally efficient labels are able to fathom unpromising labels regardless of the nature of the prolongation function, which may be nonlinear or even nonconvex, while linearity is essential for dynamic cost bounding. However, the strength of dynamic cost bounding is its speed, since it needs only one comparison test on the cost values while dominance with locally efficient labels requires the verification of $|R|+1$ inequalities. When used together, the two fathoming techniques are able to identify and discard a large percentage of the labels that correspond to ineffective partial paths. The effectiveness of the two techniques depends on the quality of the available primal information.

#### 3.2.3.2 Feasible descent directions

The motivation for this idea is the observation that at the end of each iteration we know the sequences of arcs that have contributed to the construction of feasible paths. These sequences are more likely to be part of new paths. FDDs aim to use this information efficiently to rapidly produce new paths. The costs of the previous paths are used to update the cost upper bounds at the nodes of the network. Moreover, the efficient labels that have produced these paths are also locally efficient labels, and they are therefore useful for dominating new labels. In this section we use these labels to define potential directions that may contribute to better paths.

Let $\pi \in \Pi_d$ be a feasible path generated at a given iteration of MDDPA. We use $\boldsymbol{x^\pi}$ to denote the corresponding $|A|-$solution vector. For each path $\pi \in \Pi_d$ traversing a set of nodes denoted $\mathcal{N}^\pi$, there is a set of locally efficient labels $\{l_i^\pi, i \in \mathcal{N}^\pi\}$ corresponding to the partial paths that have contributed to the construction of $\pi$. Let $\boldsymbol{x_i^\pi}$ be the $|A|-$vector corresponding to the partial path $l_i^\pi$ terminating at node $i \in \mathcal{N}^\pi$ such that $(\boldsymbol{x_i^\pi})_a = 1$ if arc $a$ is part of the partial path associated with $l_i^\pi$, and $(\boldsymbol{x_i^\pi})_a = 0$ otherwise.

**Definition 5** *Consider a feasible path $\pi$. The* direction $\boldsymbol{d_i^\pi}$ *is $\boldsymbol{d_i^\pi} = \boldsymbol{x^\pi} - \boldsymbol{x_i^\pi}$, where $i \in \mathcal{N}^\pi$.*

The directions $\boldsymbol{d_i^\pi}$ store the arcs covering all the cocycles $Co_i, Co_{i_1}, ..., Co_{|V|-1}$. These arcs can be useful for completing the extension of new labels ending at each node $i \in V$. Formally, consider a label $l_i$ at node $i \in V$, and let $\boldsymbol{x_i^l}$ be its corresponding $|A|-$vector. Let $\boldsymbol{d_i^\pi} = \boldsymbol{x^\pi} - \boldsymbol{x_i^\pi}$ be a direction induced by a previously generated path $\pi \in \Pi_d$ at node $i \in V$. The vector $\boldsymbol{x^{\pi'}} = \boldsymbol{x_i^l} + \boldsymbol{d_i^\pi}$ defines a complete path $\pi'$ from $s$ to $d$. There is no guarantee that the resulting path $\pi'$ is feasible or that it improves the cost. A test of feasibility and cost improvement is needed, and we use previously available information to check whether or not these two properties are satisfied. Figure 2 illustrates the construction of new paths using directions.

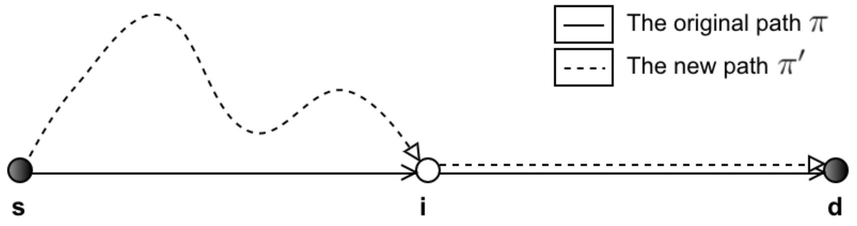

**Figure 2: Notion of direction.**

**Definition 6** *Consider a feasible path $\pi$ of cost $C_\pi$ and a direction $\boldsymbol{d_i^\pi}$ at node $i \in V$. Let $\mathcal{L}_i$ be a set of labels at node $i$. The direction $\boldsymbol{d_i^\pi}$ is said to be an FDD if there is a label $l_i \in \mathcal{L}_i$ such that the path $\pi'$ given by $\boldsymbol{x^{\pi'}} = \boldsymbol{x_i^l} + \boldsymbol{d_i^\pi}$ is feasible and $C_{\pi'} \leq C_\pi$ where $C_{\pi'}$ is the cost of $\pi'$. If we also have $C_{\pi'} \leq C^{best}$, then $\boldsymbol{d_i^\pi}$ is called an improving descent direction (IDD).*

Let $\pi$ be a feasible path and $l_i^\pi$ the locally efficient label at node $i \in V$ whose extension gave $\pi$. Consider a set of labels $\mathcal{L}_i$ at node $i$. The following proposition gives a sufficient condition that makes a direction $\boldsymbol{d_i^\pi}$ an FDD.

**Proposition 5** *Consider a label $l \in \mathcal{L}_i$ and a feasible path $\pi \in \Pi_d$. If $l$ dominates $l_i^\pi$, then $\boldsymbol{d_i^\pi} = \boldsymbol{x^\pi} - \boldsymbol{x_i^\pi}$ is an FDD.*

**Proof.** Consider a label $l = [C_l, R_l^1, R_l^2, ..., R_l^{|R|}] \in \mathcal{L}_i$ and a locally efficient label $l_i^\pi = [C_{l^\pi}, R_{l^\pi}^1, R_{l^\pi}^2, ..., R_{l^\pi}^{|R|}]$ corresponding to a feasible path $\pi$ of cost $C_\pi$. If $l$ dominates $l_i^\pi$, then $C_l \leq C_{l^\pi}$ and $R_l^k \leq R_{l^\pi}^k$ $\forall k \in$

$\{1, 2, ..., |R|\}$. Since $\pi$ is feasible, the path $\pi'$ defined by $\boldsymbol{x^{\pi'}} = \boldsymbol{x^l_i} + \boldsymbol{d^\pi_i}$ is feasible. Moreover, $C_{\pi'} = C_l + cost(\boldsymbol{d^\pi_i}) \leq C_{l^\pi} + cost(\boldsymbol{d^\pi_i}) = C_\pi$. The resulting path $\pi'$ is then a feasible path of better cost. This implies that the direction $\boldsymbol{d^\pi_i}$ is an FDD. □

In this case, the new path $\pi'$ is added to the set of available paths $\Pi_d$ to enrich the primal information. Otherwise, if the locally efficient label $l^\pi_i$ dominates $l \in \mathcal{L}_i$, the latter is not able to provide a better feasible path and can be safely eliminated from $\mathcal{L}_i$, as shown in Proposition 3.

**Remark 2** *Let $\boldsymbol{d^\pi_i}$ be an FDD and $l^\pi_i$ the locally efficient label defining it. If the path with the best cost was found using the extension of $l^\pi_i$, then $\boldsymbol{d^\pi_i}$ is an IDD.*

If no label in $\mathcal{L}_i$ dominates $l^\pi_i$, MDDPA must seek a new FDD using the extension of the sets of active labels $\mathcal{L}_i, i \in V$ by calling a DP search in the restricted search space defined by the predetermined label loading strategy. When the dominance test fails, in the sense that no label dominates another, then if the cost of one of the labels $l \in \mathcal{L}_i$ is less than the cost of $l^\pi_i$, the path $\pi'$ given by $\boldsymbol{x^{\pi'}} = \boldsymbol{x^l_i} + \boldsymbol{d^\pi_i}$ is improving, but we still need to check whether or not it is feasible.

Algorithm 5 identifies the descent directions.

**Algorithm 5:** LLEL($\mathcal{L}_i, \mathcal{P}_i, C^{best}$)

```
for all $l \in \mathcal{L}_i$ do
    for all $l' \in \mathcal{P}_i$ do
        if $l'$ dominates $l$ (label fathoming) then
            $\mathcal{L}_i \leftarrow \mathcal{L}_i \setminus \{l\}$
        if $l$ dominates $l'$ then
            if $l' \in \Pi_i$ (FDD) then
                $l^\pi_i = l'$
                $\boldsymbol{d^\pi_i} = \boldsymbol{x^\pi} - \boldsymbol{x^\pi_i}$
                $\boldsymbol{x^{\pi'}} = \boldsymbol{x^l_i} + \boldsymbol{d^\pi_i}$
                $\Pi_d \leftarrow \Pi_d \cup \pi'$
                if $C_{\pi'} < C^{best}$ (IDD) then
                    $C^{best} \leftarrow C_\pi$
                    $Cost\_Bounding(C^{best})$
```

Clearly, finding FDDs does not require significant additional computational effort. The process may quickly return paths that are able to enrich the primal information, tighten the cost upper bounds, and accordingly strengthen the label fathoming.

It is interesting to note that matching two partial paths to construct a complete path was first proposed by Righini and Salani in their bidirectional DP algorithm for the elementary SPPRC [18]. This idea was used later by Feillet et al. [11] in order to take profit from the paths related to the positive valued variables in the current solution of the MP. The MDDPA offers a different framework for the application of this idea: first, new locally efficient labels are generated at each iteration, so feasible paths are constructed dynamically using these labels; second, the feasibility is guaranteed by a simple dominance test regardless the nature of the resource constraints; third, locally efficient labels serve also to stop the extension of non promising labels.

Proposition 6 shows that MDDPA is an exact solution approach.

**Proposition 6** *MDDPA terminates by finding an optimal solution to the SPPRC.*

**Proof.** Let $\pi$ be an optimal path to the SPPRC traversing the set of nodes $\mathcal{N}^\pi$. There is a sequence of efficient labels $\{l^\pi_i, i \in \mathcal{N}^\pi\}$ corresponding to $\pi$. Using Proposition 2, there is necessarily a node $i \in V$ such that $l^\pi_i \in \mathcal{S}_i$. Suppose that $l^\pi_i$ is extended at iteration $k$ of MDDPA. The labels $\{l^\pi_j, j > i, j \in \mathcal{N}^\pi\}$ are all efficient in $G$, therefore they are locally efficient at iteration $k$. The fact that all stored labels in $\mathcal{S} = \bigcup_i \mathcal{S}_i$ are extended by MDDPA completes the proof. □

## 4 Experimentation

### 4.1 Test instances

Our test instances are derived from the well-known VCSP in urban mass transportation systems. The aim of the VCSP is to simultaneously construct bus and crew schedules that cover a set of bus trips at a minimum cost, while satisfying a set of constraints. These constraints impose the regulations of the collective agreements. A VCSP instance is defined in an acyclic network with a set of bus lines, where each line is composed of a predefined number of tasks. A task is a segment of trips that must be covered exactly once by a bus and a driver. A bus schedule is a sequence of tasks and deadheads. A deadhead is a trip without passengers that repositions the bus. A crew schedule, also called a driver schedule, is a sequence of tasks, deadheads, and breaks. The construction of these schedules is time-consuming because a driver can leave a bus line at locations called *relief points* between two consecutive tasks. The higher the number of relief points, the more difficult the problem.

The VCSP is an NP-hard problem that is solved by CG. The MP is mostly a set partitioning problem, where each task is assigned to exactly one driver and one bus, and the number of buses does not exceed the number available. Similarly, each column is associated with a possible driver schedule. The subproblems (SPs) generate schedules with negative reduced costs that satisfy the resource constraints. These are SPPRCs. For the VCSP, we have seven resource constraints.

In CG, the cost distribution on the arcs changes from iteration to iteration. The costs are reduced costs computed using the dual values of the MP constraints, and they depend on the columns present in the MP at the given iteration. For a fair comparison of MDDPA and DP we must use instances with the same reduced-cost structure. For this reason, we perform tests on SP instances extracted from some of the CG iterations.

The VCSP instances were randomly generated using the generator of [12]. These instances differ in the number of bus lines considered (120, 160, 200, 240) and the number of relief points (5,7,9); the latter is one less than the number of tasks per bus line. There are 12 possible configurations defining 12 classes of problems, denoted *rp_bl* where *rp* is the number of relief points and *bl* is the number of bus lines. For each class we generated five VCSP instances by varying the seed. These 60 instances were run in an CG solver, and for each instance, we captured two SPs, the first from the first 20% of the iterations and the second from the final 20%. We did this to evaluate our method and DP both at the beginning and at the end of the CG. This gives a total of 120 SP instances. The instances taken from the beginning of CG, called "b-instances," are indicated by "_b" in the notation of the problem class. Those taken from the end of the CG, called "e-instances," are indicated by "_e."

**Table 1: List of test instances.**

| Prob. class | # Nodes | # Arcs | # Relief points | # Tasks |
|---|---|---|---|---|
| 5_120 | 50690.0 | 78989.2 | 5 | 120 |
| 5_160 | 91458.4 | 141286.0 | 5 | 160 |
| 5_200 | 143107.2 | 220018.0 | 5 | 200 |
| 5_240 | 205377.4 | 314692.6 | 5 | 240 |
| 7_120 | 98891.2 | 152192.4 | 7 | 120 |
| 7_160 | 178498.7 | 273065.2 | 7 | 160 |
| 7_200 | 280649.0 | 427856.4 | 7 | 200 |
| 7_240 | 402557.6 | 612310.2 | 7 | 240 |
| 9_120 | 162862.0 | 249093.6 | 9 | 120 |
| 9_160 | 295783.2 | 450255.6 | 9 | 160 |
| 9_200 | 463421.0 | 703604.8 | 9 | 200 |
| 9_240 | 665201.0 | 1008197.8 | 9 | 240 |

The experiments were performed on a MacBook Pro with a 2.5 GHz processor (Intel Core i5) and 4 Gb of RAM. MDDPA was implemented in C++ using Boost, a well-known C++ library. We compare MDDPA with standard DP (std. DP) as implemented in the Boost library.

## 4.2 MDDPA vs. DP

For the 120 SP instances, MDDPA has proven itself against DP using either the NF or BF loading strategy. Tables 2 and 3 give a comparison of std. DP, MDDPA using NF, and MDDPA using BF, for the b-instances and e-instances respectively. We report for each approach the total time consumed (CPU), the number of labels created (# Lab.), and the number of calls to the dominance function (# Dom. calls). For NF and BF we give the time when we find an optimal solution for the first time (Opt. time). The results reported for each class are aggregated values of the results for the five instances.

**Table 2: Results for b_instances.**

| | std. DP | | | NF | | | | BF | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Problem class | CPU | # Lab. | # Dom. calls | CPU | Opt. time | # Lab. | # Dom. calls | CPU | Opt. time | # Lab. | # Dom. calls |
| 5_120_b | 3.56 | 7.15E+05 | 1.32E+07 | 2.21 | 1.05 | 2.54E+05 | 3.43E+06 | 2.44 | 2.15 | 4.06E+05 | 5.37E+06 |
| 5_160_b | 9.64 | 1.69E+06 | 4.23E+07 | 5.25 | 2.58 | 5.57E+05 | 1.11E+07 | 4.84 | 3.24 | 6.66E+05 | 1.10E+07 |
| 5_200_b | 21.53 | 3.29E+06 | 1.09E+08 | 10.93 | 6.67 | 1.21E+06 | 4.17E+07 | 13.59 | 11.97 | 1.65E+06 | 5.82E+07 |
| 5_240_b | 33.65 | 5.56E+06 | 1.86E+08 | 21.01 | 16.42 | 2.51E+06 | 7.78E+07 | 24.97 | 10.52 | 3.12E+06 | 1.05E+08 |
| 7_120_b | 8.84 | 1.63E+06 | 3.80E+07 | 4.76 | 3.22 | 5.37E+05 | 8.08E+06 | 6.27 | 4.59 | 9.24E+05 | 1.41E+07 |
| 7_160_b | 18.93 | 3.63E+06 | 1.00E+08 | 15.30 | 10.63 | 1.92E+06 | 5.76E+07 | 14.54 | 8.79 | 2.01E+06 | 5.45E+07 |
| 7_200_b | 51.39 | 7.81E+06 | 3.07E+08 | 33.90 | 20.76 | 3.66E+06 | 1.62E+08 | 47.80 | 34.74 | 5.13E+06 | 2.67E+08 |
| 7_240_b | 89.38 | 1.33E+07 | 5.62E+08 | 53.45 | 30.35 | 5.26E+06 | 2.24E+08 | 62.83 | 29.92 | 6.68E+06 | 2.99E+08 |
| 9_120_b | 15.29 | 3.02E+06 | 7.65E+07 | 9.71 | 5.97 | 1.09E+06 | 2.20E+07 | 10.97 | 8.04 | 1.51E+06 | 2.50E+07 |
| 9_160_b | 41.98 | 7.24E+06 | 2.35E+08 | 24.95 | 14.99 | 2.53E+06 | 6.70E+07 | 22.73 | 9.85 | 2.65E+06 | 6.98E+07 |
| 9_200_b | 93.57 | 1.41E+07 | 5.94E+08 | 70.68 | 46.90 | 6.64E+06 | 2.80E+08 | 86.90 | 44.49 | 8.06E+06 | 4.72E+08 |
| 9_240_b | 176.64 | 2.37E+07 | 1.09E+09 | 114.91 | 61.69 | 1.09E+07 | 5.41E+08 | 132.42 | 19.63 | 1.24E+07 | 7.28E+08 |

**Table 3: Results for e_instances.**

| | std. DP | | | NF | | | | BF | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Problem | CPU | # Lab. | # Dom. calls | CPU | Opt. time | # Lab. | # Dom. calls | CPU | Opt. time | # Lab. | # Dom. calls |
| 5_120_e | 1.18 | 2.81E+05 | 1.49E+06 | 0.17 | 0.17 | 2.37E+03 | 1.36E+04 | 0.19 | 0.19 | 2.37E+03 | 1.36E+04 |
| 5_160_e | 3.37 | 7.64E+05 | 7.42E+06 | 1.38 | 0.49 | 1.25E+05 | 1.66E+06 | 1.55 | 0.50 | 1.39E+05 | 1.73E+06 |
| 5_200_e | 6.97 | 1.53E+06 | 2.20E+07 | 4.05 | 1.26 | 4.62E+05 | 9.19E+06 | 5.13 | 2.10 | 6.27E+05 | 1.14E+07 |
| 5_240_e | 12.17 | 2.43E+06 | 3.89E+07 | 3.91 | 2.34 | 2.79E+05 | 3.73E+06 | 4.14 | 2.19 | 2.98E+05 | 4.24E+06 |
| 7_120_e | 2.32 | 5.43E+05 | 2.74E+06 | 0.34 | 0.34 | 3.15E+03 | 1.39E+04 | 0.34 | 0.34 | 3.15E+03 | 1.39E+04 |
| 7_160_e | 8.77 | 1.84E+06 | 2.44E+07 | 2.36 | 0.95 | 2.28E+05 | 2.50E+06 | 2.55 | 0.87 | 2.61E+05 | 3.07E+06 |
| 7_200_e | 18.17 | 3.75E+06 | 6.45E+07 | 6.14 | 3.45 | 6.07E+05 | 1.28E+07 | 7.88 | 4.46 | 9.77E+05 | 1.76E+07 |
| 7_240_e | 26.89 | 5.18E+06 | 9.07E+07 | 8.98 | 6.53 | 6.36E+05 | 9.04E+06 | 7.80 | 5.20 | 5.79E+05 | 7.92E+06 |
| 9_120_e | 5.69 | 1.26E+06 | 1.11E+07 | 1.48 | 0.69 | 1.04E+05 | 5.54E+05 | 2.33 | 1.73 | 2.46E+05 | 1.60E+06 |
| 9_160_e | 16.19 | 3.29E+06 | 4.42E+07 | 7.10 | 4.13 | 6.64E+05 | 8.51E+06 | 7.77 | 4.70 | 9.19E+05 | 1.30E+07 |
| 9_200_e | 42.38 | 7.67E+06 | 2.13E+08 | 12.83 | 4.41 | 1.20E+06 | 1.51E+07 | 13.00 | 5.17 | 1.32E+06 | 2.24E+07 |
| 9_240_e | 56.54 | 1.07E+07 | 2.18E+08 | 18.20 | 17.76 | 1.31E+06 | 2.32E+07 | 16.43 | 13.34 | 1.19E+06 | 1.84E+07 |

In terms of solution time, Tables 2 and 3 show that MDDPA outperforms std. DP at both the beginning and the end of the CG. Figure 3 presents the time reduction factors obtained as a ratio of std. DP time to NF and BF time. It shows that the factors achieved by the two loading strategies are significant, especially for the e-instances. The BF strategy has a reduction factor for the b-instances ranging between 1.08 and 1.99 with an average of 1.44. This ratio is greater for the e-instances: up to 6.81, with an average of 3.33. The NF strategy is better. For b-instances, the total time is reduced by a factor varying between 1.24 and 1.97, with an average of 1.62. For e-instances, it is up to 6.87, with an average of 3.59. Section 4.3 compares NF and BF.

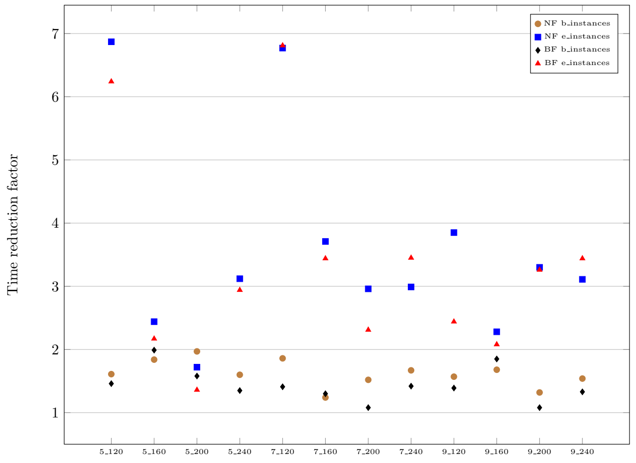

**Figure 3: Time reduction factor (MDDPA vs. std. DP).**

The time reductions achieved by MDDPA are induced by huge reductions in the number of labels. For b-instances, NF and BF give an overall label reduction factor of 2.24 on average; for e-instances this factor is on average 30.25 for NF and 29.18 for BF. A direct consequence is a reduction in the number of dominance calls. For b-instances, the dominance reduction factor ranges between 1.15 and 3.84 for BF, while it is up to 4.71 with an average of 2.89 for NF. For e-instances this factor is on average 31.46 for BF and 33.18 for NF. These reductions are a result of the quality of the labels generated during the first iterations of MDDPA. This primal information is efficiently used by the dynamic cost bounding and dominance with locally efficient labels. Many labels are fathomed, greatly reducing the time spent on dominance tests.

## 4.3 NF vs. BF

Tables 4 and 5 give a comparison of NF and BF, for the b-instances and e-instances respectively. We report, for each problem class and each loading strategy, the averages of: the number of iterations (# Its.); the number of descent directions found during the solution process (#FDD); the percentage of the cost decrease attributed to these FDDs (% ACD by FDD) with respect to the cost of the previous paths from which the directions are constructed; the percentage of the cost improvement attributed to IDDs (% Imp. of IDD); and the percentage of the std. DP time required for MDDPA to find an optimal solution (% Time in opt.). For the b-instances, we give the percentage of the optimal value returned by MDDPA in less than 10% of the std. DP time (% Opt. in 10%). This sheds light on one of the most interesting feature of MDDPA: its ability to quickly find good solutions, especially in the first CG iterations.

**Table 4: MDDPA for b_instances.**

| | NF | | | | | | BF | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Problem class | # Its. | # FDD | % ACD by FDD | % Imp. of IDD | % Time in opt. | % Opt. in 10% | # Its. | # FDD | % ACD by FDD | % Imp. of IDD | % Time in opt. | % Opt. in 10% |
| 5_120_b | 8.0 | 17.2 | 26.0 | 15.7 | 29.6 | 82.2 | 3.0 | 27.8 | 20.1 | 7.9 | 60.5 | 46.9 |
| 5_160_b | 8.0 | 62.0 | 21.6 | 13.0 | 26.8 | 85.4 | 3.6 | 356.0 | 16.3 | 4.8 | 33.6 | 59.3 |
| 5_200_b | 7.8 | 255.3 | 15.1 | 23.2 | 31.0 | 88.4 | 4.3 | 2061.3 | 9.4 | 15.0 | 55.6 | 50.0 |
| 5_240_b | 8.0 | 366.6 | 15.7 | 22.7 | 48.8 | 82.6 | 5.0 | 1830.0 | 10.9 | 14.6 | 31.3 | 50.8 |
| 7_120_b | 7.6 | 71.0 | 24.1 | 10.1 | 36.4 | 75.6 | 4.0 | 261.0 | 19.5 | 5.5 | 51.9 | 45.1 |
| 7_160_b | 8.0 | 362.5 | 11.5 | 3.3 | 56.2 | 81.6 | 4.0 | 324.2 | 13.1 | 2.9 | 46.4 | 66.6 |
| 7_200_b | 7.8 | 282.3 | 15.2 | 17.6 | 54.0 | 86.0 | 5.0 | 2560.3 | 10.9 | 21.5 | 52.9 | 54.5 |
| 7_240_b | 8.0 | 538.0 | 9.4 | 9.6 | 36.3 | 80.5 | 5.3 | 4556.5 | 8.1 | 8.1 | 25.3 | 59.5 |
| 9_120_b | 7.6 | 149.4 | 23.8 | 13.8 | 39.1 | 81.9 | 4.0 | 79.6 | 18.0 | 10.5 | 52.6 | 56.4 |
| 9_160_b | 8.0 | 247.4 | 23.7 | 13.3 | 35.7 | 79.0 | 4.4 | 1067.0 | 27.2 | 18.9 | 23.5 | 64.7 |
| 9_200_b | 8.0 | 648.0 | 13.7 | 15.9 | 39.4 | 92.8 | 5.7 | 3843.0 | 7.7 | 16.2 | 33.8 | 56.5 |
| 9_240_b | 8.0 | 698.8 | 13.4 | 18.3 | 34.9 | 85.0 | 6.0 | 4170.8 | 10.1 | 18.7 | 11.1 | 69.6 |

**Table 5: MDDPA for e_instances.**

| Problem class | NF # Its. | NF # FDD | NF % ACD by FDD | NF % Imp. of IDD | NF % Time in opt. | BF # Its. | BF # FDD | BF % ACD by FDD | BF % Imp. of IDD | BF % Time in opt. |
|---|---|---|---|---|---|---|---|---|---|---|
| 5_120_e | 1.0 | 0.0 | 0.0 | 0.0 | 14.6 | 1.0 | 0.0 | 0.0 | 0.0 | 16.0 |
| 5_160_e | 3.6 | 1.4 | 21.1 | 5.8 | 14.6 | 3.2 | 1.4 | 21.1 | 5.8 | 14.8 |
| 5_200_e | 4.0 | 36.3 | 34.7 | 13.9 | 18.1 | 4.0 | 41.3 | 33.1 | 21.2 | 30.2 |
| 5_240_e | 3.4 | 38.8 | 30.2 | 8.6 | 19.2 | 3.2 | 29.2 | 33.0 | 5.7 | 18.0 |
| 7_120_e | 1.0 | 0.0 | 0.0 | 0.0 | 14.8 | 1.0 | 0.0 | 0.0 | 0.0 | 14.7 |
| 7_160_e | 2.6 | 57.0 | 13.7 | 0.0 | 10.8 | 2.6 | 56.6 | 14.7 | 0.8 | 9.9 |
| 7_200_e | 3.8 | 108.2 | 34.8 | 10.7 | 19.0 | 3.6 | 121.0 | 35.9 | 10.7 | 24.6 |
| 7_240_e | 3.4 | 44.0 | 19.5 | 14.0 | 24.3 | 3.2 | 44.8 | 19.2 | 9.3 | 19.3 |
| 9_120_e | 2.6 | 9.4 | 49.4 | 35.3 | 12.1 | 2.4 | 4.6 | 48.8 | 35.3 | 30.5 |
| 9_160_e | 3.4 | 40.8 | 18.5 | 11.9 | 25.5 | 3.2 | 8.2 | 22.5 | 11.9 | 29.0 |
| 9_200_e | 4.0 | 156.6 | 35.3 | 41.6 | 10.4 | 3.4 | 144.4 | 28.7 | 20.5 | 12.2 |
| 9_240_e | 4.0 | 180.0 | 35.5 | 6.5 | 31.4 | 4.0 | 203.5 | 47.4 | 19.0 | 23.6 |

Tables 4 and 5 indicate the excellent quality of solutions returned by NF and BF in a small proportion of the time consumed by DP. For the b-instances, NF returns an optimal solution in 26.77% to 56.18% of the std. DP time, with an average of 39.01%. BF requires 11.11% to 60.48% of the std. DP time, with approximately the same average. BF finds the optimal solution faster than NF does, especially for instances with more than 160 bus lines and 7 or 9 relief points. For the e-instances, the average time to find an optimal solution is 17.89% for NF and 20.23% for BF.

For the b-instances, in less than 10% of the std. DP time, NF finds paths with a cost ranging between 75.65% and 92.82% of the optimal value, and for BF the range is 45.11% to 69.56%. Figures 4 and 5 track the evolution of the objective value over time for NF and BF on the largest b-instance and e-instance respectively.

These results are useful in a CG context. The SPs aim to quickly provide the MP with feasible paths (columns) with good reduced costs that are not necessarily optimal. This is especially important in the first CG iterations, where the dual values are not yet stable, and the goal is to quickly feed the MP with more columns to stabilize the dual-value distribution. The need to prove optimality arises only during the final iterations.

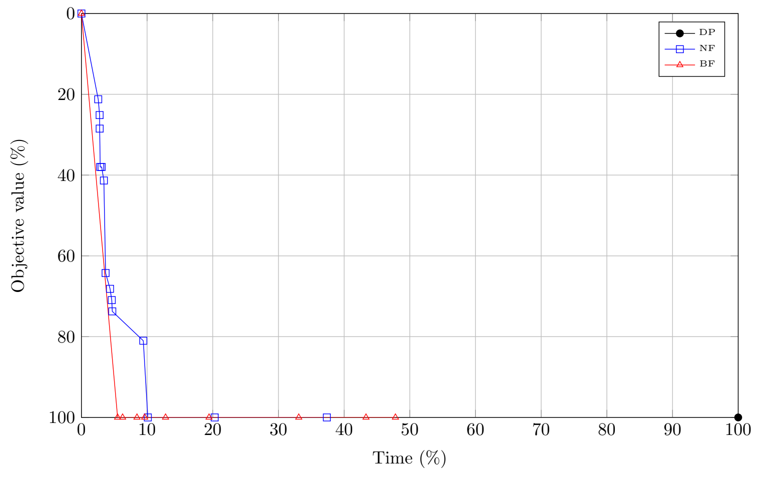

**Figure 4: Improvement in objective value as function of time for b_instances.**

We wish to highlight the impact of the FDDs and IDDs on NF and BF; these directions are used to construct better feasible paths. FDDs and IDDs are found for all the b-instances. The number of paths generated by FDDs is on average 291.14 for NF and 1663.25 for BF. For e-instances, the number of FDDs is 56 on average for NF and BF. FDDs are thus more effective in the b-instances, especially with BF. This may be because at the end of the CG process, the variation in the reduced costs on the arcs becomes small. Consequently, it is harder to find directions that are able to improve the previously generated paths.

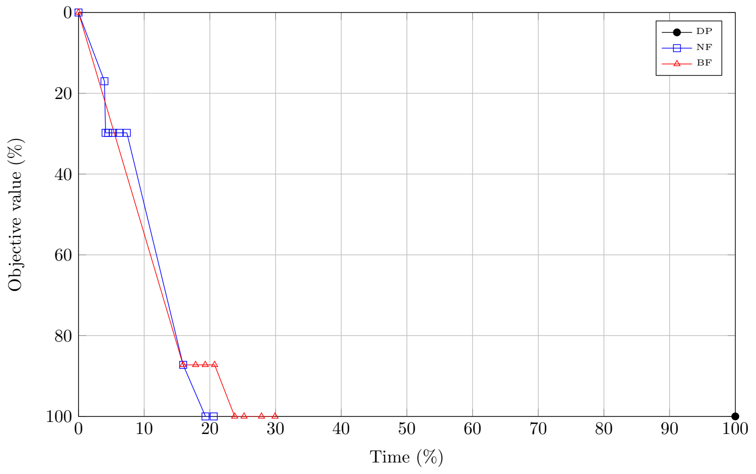

**Figure 5: Improvement in objective value as function of time for e_instances.**

Moreover, in e-instances the SPs are tighter, so the number of paths with negative reduced costs is smaller, and consequently the probability of finding such paths decreases.

IDDs improve the cost by 14.65% on average for b-instances and 12.35% for e-instances. These values are slightly lower for BF. Figure 6 shows the impact of these directions on MDDPA, for one large instance from class 9_240. Here an optimal solution is found in less than 10% of the total time when IDDs are used, and in 25% of the total time when IDDs are not used. We conclude that IDDs are able to considerably decrease the objective value with just a little additional computational effort, and to tighten the cost upper bounds, reducing the number of labels and improving the total time.

MDDPA outperforms the std. DP for all the instances. It reduces the overall solution time by a factor of up to 3.5. Furthermore, as a primal method, it can generate interesting feasible paths in a limited solution time. It is expected to lead to a huge improvements in the overall CG solution time. NF can quickly make small improvements to the objective value, and it proves optimality earlier than BF does. However, BF converges more quickly to an optimal solution, especially for the most difficult b-instances. To summarize, either strategy can be used in the first CG iterations depending on the quality of columns deemed sufficient by the modeler, and NF is more appropriate for the final iterations, when we must prove optimality.

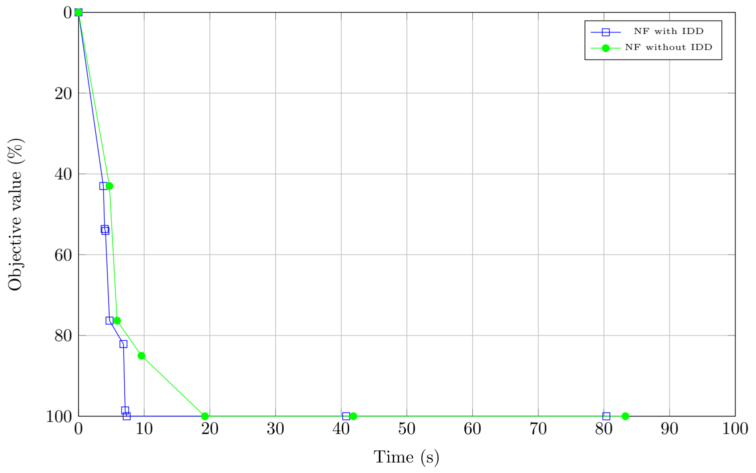

**Figure 6: Effect of IDDs on MDDPA.**

## 5 Conclusion

We have presented a new solution approach for the SPPRC. MDDPA combines three techniques that work together to deal with the problem of dimensionality and to overcome the weaknesses of DP algorithms. The algorithm performs a disjoint partition of the search space using a label storing procedure. It then uses two loading strategies for the iterative exploration of a sequence of restricted search spaces. These two techniques enable the construction of a primal framework that allows the current iteration to take advantage of previous ones. In particular, our label fathoming techniques discard a huge number of labels. Also, the LLEL procedure allows the detection of directions that help to provide the solver with good new paths.

We evaluated the algorithm on instances of the simultaneous VCSP, extracted from different stages of the CG process. MDDPA performs considerably better than std. DP, especially in the final CG iterations. Furthermore, it provides fast convergence to the optimal solution. For b-instances, more than 80% of the optimal solution is found in less than 10% of the time required by DP.

In the CG context, MDDPA has two main advantages. First, as a primal method, it provides a sequence of columns of nonincreasing cost, and it allows premature stopping of the solution process when the quality of the columns is sufficient. Second, it offers two loading strategies, allowing different strategies to be used at different stages of the CG. Future research will focus on embedding the MDDPA within a CG scheme and using it in different applications.

## References

[1] Beasley, J.E., Christofides, N. An algorithm for the resource constrained shortest path problem. Networks 19(4), 379–394 (1989).

[2] Carlyle, W.N., Royset, J.O., Wood, R.K. Lagrangean relaxation and enumeration for solving constrained shortest-path problems. Networks 52, 256–270 (2008).

[3] Desaulniers, G. et al. Crew pairing at Air France. European Journal of Operational Research 97(2), 245–259 (1997).

[4] Desaulniers, G., Villeneuve, D. The shortest path problem with time windows and linear waiting costs. Transportation Science 34(3), 312–319 (2000).

[5] Desrochers, M. La fabrication d'horaires de travail pour les conducteurs d'autobus par une mthode de gnration de colonnes. Ph.D. thesis (1986), Universit de Montral, Montreal, Canada.

[6] Desrochers, M., Soumis, F. A generalized permanent labeling algorithm for the shortest path problem with time windows. INFOR 26, 191–212 (1988).

[7] Desrochers, M., Soumis, F. A reoptimization algorithm for the shortest path problem with time windows. European Journal of Operational Research 35, 242–254 (1988).

[8] Desrosiers, J., Dumas, Y., Solomon, M.M., Soumis, F. Time constrained routing and scheduling. In: Ball, M.O. et al. (Eds.), Network Routing, Handbook in Operations Research and Management Science 8. Elsevier Science, Amsterdam, 35–139 (1995).

[9] Di Puglia Pugliese, L., Guerriero, F. A survey of resource constrained shortest path problems: Exact solution approaches. Networks 62(3), 183–200 (2013).

[10] Dumitrescu, I., Boland, N. Improved preprocessing, labeling and scaling algorithm for the weight-constrained shortest path problem. Networks 42(3), 135–153 (2003).

[11] Feillet, D., Gendreau, M., Rousseau, L.-M. New refinements for the solution of vehicle routing problems with branch & price. INFOR 45(4), 239–256 (2007).

[12] Haase, K., Desaulniers, G., Desrosiers, J. Simultaneous vehicle & crew scheduling in urban mass transit systems. Transportation Science 35, 286–303 (2001).

[13] Halpern, J., Priess, J. Shortest paths with time constraints on moving and parking. Networks 4(3), 241–253 (1974).

[14] Handler, G.Y., Zang, I. A dual algorithm for the constrained shortest path problem. Networks 10, 293–309 (1980).

[15] Lozano, L., L. Medaglia, A. On an exact method for the constrained shortest path problem. Computers and Operations Research, 40(1), 378–384 (2013).

[16] Melhorn, K., Ziegelmann, M. Resource constrained shortest paths. 7th Annual European Symposium on Algorithms (ESA 2000), Lecture Notes in Computer Science 1879. Springer-Verlag, Berlin, Heidelberg, 326–337 (2000).

[17] Nagih, A., Soumis, F. Nodal aggregation of resource constraints in shortest path problem. EJOR 172(2), 500–514 (2006).

[18] Righini, G., Salani, M. Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. Discrete Optimization 3, 255–273 (2006).

[19] Santos, L., Coutinho-Rodrigues, J., Current, J.R. An improved solution algorithm for the constrained shortest path problem. Transportation Res. Part B 41(7), 756–771 (2007).

[20] Xue, G. Primal-dual algorithms for computing weight-constrained shortest paths and weight-constrained minimum spanning trees. Proc. $19^{th}$ IEEE Internat. Performance, Comput., Comm. Conf. (IPCCC), 271–277 (2000).

[21] Zabarankin, M., Uryasev, S., Pardalos, P. Optimal risk path algorithms. Murphey, R., Pardalos, P., eds. Cooperative Control and Optimization. Kluwer, Dordrecht, the Netherlands, 271–303 (2001).