Les Cahiers du GERAD ISSN: 0711–2440

A primal adjacency-based algorithm for the shortest path problem with resource constraints

I. Himmich, H. Ben Amor
I. El Hallaoui, F. Soumis

G–2018–09

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

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

A primal adjacency-based algorithm for the shortest path problem with resource constraints

Ilyas Himmich [a,b]

Hatem Ben Amor [a,c]

Issmail El Hallaoui [a,b]

François Soumis [a,b]

[a] *GERAD HEC Montréal, Montréal (Québec), Canada, H3T 2A7*

[b] *Department of Mathematics and Industrial Engineering, Polytechnique Montréal, Montréal (Québec), Canada, H3C 3A7*

[c] *AD OPT, a Kronos Division, Montréal (Québec), Canada, H3V 1H8*

ilyas.himmich@gerad.ca
issmail.elhallaoui@gerad.ca
hatem.ben.amor@gerad.ca
franois.soumis@gerad.ca

February 2018
Les Cahiers du GERAD
G–2018–09

Abstract: The shortest path problem with resource constraints (SPPRC) is often used as a subproblem within a column generation approach for routing and scheduling problems. It aims to find a least cost path between the source and the destination nodes in a network while satisfying the resource consumption limitations on every node. The SPPRC is usually solved using dynamic programming. Such approaches are effective in practice, but they can be inefficient when the network is large and especially when the number of resources is high. To cope with this major drawback, we propose a new exact primal algorithm that explores the solution space iteratively using a path-adjacency-based partition. Numerical experiments for vehicle and crew scheduling instances demonstrate that the new approach outperforms both the standard dynamic programming and the multi-directional dynamic programming methods.

Keywords: Shortest path problem with resource constraints, column generation, adjacency, dynamic programming

1 Introduction

One of the most important problems in the context of large-scale transportation models is the shortest path problem with resource constraints (SPPRC). It was introduced by Desrochers [4] as an extension of the shortest path problem with time windows. It aims to find the least cost path between the source and the destination nodes in a network, while satisfying the resource consumption limits on every node. The SPPRC is often used as a subproblem in the solution of vehicle routing and crew scheduling problems by column generation (CG). In the airline industry, crew rostering requires at each iteration the solution of hundreds or thousands of subproblems, one for each crew member. Resources are used to model the collective agreement and safety rules as well as other aspects related to the solution quality. A few dozen resources are generally used.

We propose in this work a new exact primal adjacency-based (PAB) algorithm that explores the solution space iteratively using a path-adjacency-based partition. Our study will focus on acyclic networks, which are often used in vehicle and crew scheduling applications in the airline industry [3, 18].

1.1 Literature review on exact solution methods

The SPPRC has been widely studied, and several exact and heuristic algorithms have been proposed. They can be classified into three main groups (see [8]): 1) path ranking methods, 2) branch-and-bound (B&B) based methods, and 3) dynamic programming (DP) methods. Path ranking methods sort the paths in the network by increasing cost until a feasible one is found. The first path ranking algorithm was proposed by Handler and Zang [13], and it was improved in [20]. The major drawback of this approach is that the number of paths that must be ranked increases exponentially with the size of the network. This makes the method computationally intractable.

B&B approaches have been proposed by Beasley and Christofides [1], Carlyle et al. [2], and Muhandiramge and Boland [17]. Each node in the B&B tree represents a subpath from the source node s to a given node in the network. The nodes for infeasible subpaths are removed from the tree, and upper and lower bounds are used to prune unpromising nodes. Thus, the efficiency depends on the quality of these bounds.

In the context of CG, DP has proved to be the most effective method in practice [19]. The basic DP algorithm [5] for a simplified version of the SPPRC is an extension of the Bellman–Ford algorithm. The algorithm assigns *labels* to each node i. The label is a multidimensional vector storing the cost and the total resource consumption of a subpath from the source node s to node i. Label dominance rules as well as pruning strategies are used to omit unpromising subpaths, i.e., subpaths that cannot be used to produce optimal paths. This approach can handle complex rules that may be nonlinear and even nonconvex, and it can provide the CG master problem with several columns at each iteration.

Several researchers have developed efficient DP algorithms for the SPPRC, e.g., Desrosiers et al. [7] and Desrochers and Soumis [5, 6]. An improved version of the labeling algorithm described in [5] was proposed by Dumitrescu and Boland [10]. They use information from the solution of a Lagrangean dual problem to compute lower and upper bounds. This prunes more labels, which reduces the solution time. Muhandiramge and Boland [17] introduced preprocessing techniques to reduce the size of the network. New refinements of the solution of the SPPRC inside CG process, were proposed by Feillet et al. [11]. One of these refinements is a limited discrepancy search algorithm that prioritizes the most promising arcs when extending labels. Another exact solution was introduced by Lozano et al. [16], it relies on implicit enumeration of feasible paths, and uses some pruning schemes to narrows the search space.

Recently, Himmich et al. [14] have proposed a new multi-directional dynamic programming algorithm (MDDPA) for the SPPRC. This approach is a generalization of the classical mono-directional DP algorithm. It stores, in an initialization step, sets of efficient labels at different nodes of the network, before extending these labels sequentially in several iterations according to a predetermined search strategy. MDDPA provides two search strategies: Nearest First first extends the labels closest to the destination node, and Best First prioritizes the extension of the labels with the most negative reduced costs. This algorithm performed well compared to the standard DP method.

1.2 Approximate algorithms for column generation

Solving large instances of the SPPRC with DP is time-consuming, especially when the set of rules is large and complex. In fact, as the number of resources increases, the number of labels becomes excessive for large networks. Many approximate methods have been developed to cope with this major drawback. In a CG context, we need to solve the SPPRC to proven optimality only in the last iteration, to show that there are no more negative reduced cost paths. Approximate algorithms suffice in the preceding iterations. These techniques are based on limiting the number of labels to retain at a given node; extending the labels on a subset of arcs, or testing dominance on a preselected subset of resources [15]. These techniques drastically reduce the number of labels at each iteration, but the solution is likely suboptimal, and the process may be time-consuming. In addition, the choice of the labels to keep, the arcs to use, and the resources to dominate on is based on the intuition and experience of the planners rather than mathematical rules, so the algorithm requires the adjustment of several parameters for each problem and potentially each instance. Moreover, this tuning must be performed many times during the solution process, which is time-consuming.

Nagih and Soumis [18] proposed a different approach to mitigate the impact of the large number of resources. They project the resource vector onto a lower dimensional subspace. They define a Lagrangean dual problem by relaxing a subset of the resource constraints, and the corresponding Lagrangean multipliers are the coefficients of the projection matrix. The experiments show that their relaxation must be modified at each iteration to generate adequate negative reduced cost paths.

1.3 Contributions and organization

The contributions of this paper are fivefold:

i. We present a new polyhedral study that allows us to split the search space of the SPPRC into small disjoint subspaces. This partition is defined using path-adjacency-based nested neighborhoods.

ii. We develop a new primal exact algorithm that explores iteratively these subspaces using an improved version of DP without regeneration of labels, until optimality is proved.

iii. Unlike the existing methods, including MDDPA, the PAB algorithm benefits from available primal information, e.g., previous schedules, to build a good starting solution. This primal information is enriched at the subsequent iterations by the most promising paths generated during the solution process.

iv. Being a primal method, the PAB algorithm is able to produce sets of feasible paths of nonincreasing cost at each iteration. We use these paths for two purposes: First, to tighten the cost bounds at each node of the network, which reduces the size of the search space in the subsequent iterations. Second, to construct a good starting point for a subsequent iteration using affine combinations of the previously generated paths. This greatly reduces the number of labels generated and thus accelerates the solution process.

v. Our extensive experiments show that the PAB algorithm is better than the state-of-the-art DP algorithms on large-scale acyclic network instances with up to 600.000 nodes and 1.000.000 arcs. It returns very interesting solutions in very limited portions of time, and drastically reduces the number of created labels. These results show that the proposed approach provides a highly efficient solution framework, nicely suitable for CG method.

The rest of the paper is organized as follows. Section 2 formally defines the SPPRC using the cocycle-based formulation introduced in [14], and Section 3 presents a polyhedral study where path-adjacency in networks is defined and linked to the notion of a *detour*. The new algorithm is presented in Section 4, and its efficiency is demonstrated in Section 5 where an extensive numerical study is discussed. Section 6 provides concluding remarks.

2 Mathematical formulation

Consider an acyclic network $G(V, A)$ where V is the set of nodes including the source node s and the destination node d, and $A = \{(i, j)/\ i, j \in V\}$ is the set of arcs. The SPPRC computes the minimum-cost

path among all feasible paths starting from s and ending at d. A path is said to be feasible if it respects the resource constraints induced by a set of resources R.

The resource constraints are based on resource consumptions and resource intervals. The resource consumptions are $|R|$-dimensional vectors $(r_{ij}^1, r_{ij}^2, ..., r_{ij}^{|R|})$ associated with each arc $(i,j) \in A$ where r_{ij}^t is the quantity of resource $t \in R$ consumed when traversing arc (i,j). The resource intervals, also called resource windows, are a set of intervals $[a_i^t, b_i^t]$ associated with each node $i \in V$, where a_i^t and b_i^t are the lower and upper bounds on the resource $t \in R$ consumed along an s-i subpath.

DP algorithms associate with each s-i subpath π_i an $(|R|+1)$ vector $(C_i, R_i^1, R_i^2, ..., R_i^{|R|})$ called a *label*. C_i and R_i^t denote respectively the cost and the consumption of each resource $t \in R$ over all the arcs composing π_i. Hence, π_i is feasible if $a_i^t \leq R_i^t \leq b_i^t$, $\forall t \in R$.

In addition to the resource constraints, path-structure constraints are used to ensure the connectivity of the path. These constraints are basically flow conservation constraints that allow the circulation of one unit of flow from s to d. Additional path-structure constraints may be added to the formulation depending on the requirements of the problem; there may be forbidden paths, or a need for elementary paths in cyclic networks. Hence, the SPPRC is formulated as follows:

$$(P_1) \quad \text{Minimize} \quad \sum_{(i,j)\in A} c_{ij} x_{ij} \tag{2.1}$$

$$\sum_{i\in V} x_{ij} - \sum_{i\in V} x_{ji} = \begin{cases} -1 & \text{if } j = s \\ 0 & \text{if } j \in V \setminus \{s,d\} \\ 1 & \text{if } j = d \end{cases} \tag{2.2}$$

$$x_{ij}(R_i^t + r_{ij}^t - R_j^t) \leq 0 \qquad \forall t \in R, \forall (i,j) \in A \tag{2.3}$$

$$a_i^t \leq R_i^t \leq b_i^t \qquad \forall t \in R, \forall i \in V \tag{2.4}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A \tag{2.5}$$

Constraints (2.2) are the flow conservation constraints. Constraints (2.3) model the resource consumptions whenever an arc (i,j) is chosen to be a part of the solution (path), while constraints (2.4) require the resource consumption along each s-i subpath 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 (2.5) are the binary requirements on the arc variables x_{ij}, $(i,j) \in A$.

The flow conservation constraints (2.2) were reformulated by Himmich et al. [14] using the notion of a *cocycle*. They first sort the nodes of $G(V,A)$ in topological order. Each node is denoted by its rank in that order, and the source and destination nodes are indexed respectively by 1 and $|V|$. A cocycle is defined as follows.

Definition 1 *([14]) Let $G(V,A)$ be an acyclic network, where $V = \{1,2,...,|V|\}$ is topologically ordered, and consider a node $k \in V$. The k^{th} cocycle, denoted Co_k, is the set of all arcs $(i,j) \in A$ such that the origin $i \in V$ is ordered before node k and the destination j is ordered strictly after node k. Formally, $Co_k = \{(i,j) \in A | i \leq k < j\}$.*

The authors associate with each cocycle a constraint called a *cocycle constraint*. These constraints are as follows:

$$\sum_{(i,j)\in Co_k} x_{ij} = 1 \qquad \forall k \in \{1,2,...,|V|-1\}. \tag{2.6}$$

Figure 1, reproduced from [14], shows the cocycle constraints for a four-node acyclic network.

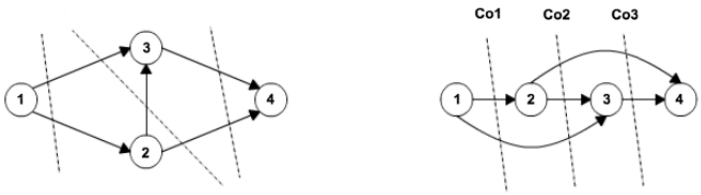

Figure 1: Cocycle constraints.

Himmich et al. [14] proved that the cocycle constraints are equivalent to the flow conservation constraints in acyclic connected networks. Hence, they reformulate the SPPRC as a set partitioning problem with side constraints where each column represents an arc covering a subset of the cocycles. A solution to the problem is a feasible path covering all cocycles exactly once. The new formulation is as follows:

$$\begin{array}{lll} (P_2) & \text{Minimize} & \boldsymbol{c}^T\boldsymbol{x} \\ & \boldsymbol{M}\boldsymbol{x} = \boldsymbol{e} & \\ & (2.3) - (2.5) & \end{array}$$

where $\boldsymbol{M} = [m_{ka}]_{(|V|-1).|A|}$ such that $k \in V \setminus \{d\}, a \in A$ is a $(|V|-1) \times |A|$ matrix such that $m_{ka} = 1$ if arc a covers cocycle Co_k and 0 otherwise; $\boldsymbol{c}$ is the arc cost vector; and $\boldsymbol{e} = (1, ..., 1)^T$ is a (—V—-1) vector.

Corollary 1 *The matrix $\boldsymbol{M}$ is full rank.*

Proof. Since the network G is connected, for each cocycle Co_k, there is at least one arc a such that $m_{ka} = 1$ and $m_{la} = 0\ \forall l < k$. The columns corresponding to these arcs can then be rearranged to form a triangular submatrix $\boldsymbol{N}$ such that $\boldsymbol{N}_{kk} = 1$ and $\boldsymbol{N}_{kl} = 0$, $\forall k \in \{1, 2, ..., |V|-1\}, l < k$. Consequently, the $|V|-1$ rows are linearly independent, which completes the proof. □

In what follows, we denote by $\mathcal{P}^{\mathcal{SPP}}$ the polyhedron for the ordinary shortest path problem (obtained by relaxing the resource constraints (2.3) and (2.4) and the integrality constraints (2.5)).

Corollary 2 *Since the $|V|-1$ cocycle constraints are linearly independent, the dimension of $\mathcal{P}^{\mathcal{SPP}}$ is $|A| - |V| + 1$.*

3 Polyhedral study

We now study some polyhedral properties related to path adjacency in $\mathcal{P}^{\mathcal{SPP}}$. We discuss adjacency to a single path in Section 3.1, and we generalize this to a set of paths in Section 3.2. Based on the path adjacency, we derive the notion of degree of adjacency, which we will use to classify the feasible solutions to SPPRC. This classification is used to partition the solution space of the SPPRC into several disjoint subspaces that will be explored by the PAB algorithm (in the next section).

We decided to focus on $\mathcal{P}^{\mathcal{SPP}}$ instead of the solution space of the SPPRC for three reasons. First, all the solutions of the SPPRC are paths from s to d, and their corresponding extreme points are all integers. Therefore, every feasible extreme point in the solution space of the SPPRC is also an extreme point of $\mathcal{P}^{\mathcal{SPP}}$. Second, the adjacency between two feasible extreme points in $\mathcal{P}^{\mathcal{SPP}}$ remains valid in the convex hull of the solutions of SPPRC. Third, every partition of the set of solutions in $\mathcal{P}^{\mathcal{SPP}}$ induces a partition in the solution space of the SPPRC. Thus, if we consider in addition that the solution space of the SPPRC is not convex in general, and does not define a polyhedron, we assume that $\mathcal{P}^{\mathcal{SPP}}$ allows the extraction of polyhedral properties describing the solutions of SPPRC. Obviously, $\mathcal{P}^{\mathcal{SPP}}$ has been widely studied in both graph theory and linear optimization, but to the best of our knowledge, the results we present are new, and this is the first work to use the notion of adjacency to solve the SPPRC.

3.1 Adjacency to a single path

As seen in Section 2, the ordinary shortest path problem can be equivalently formulated as the following set partitioning problem:

$$(P_3) \quad \text{Minimize} \quad \boldsymbol{c}^T\boldsymbol{x}$$
$$\boldsymbol{M}\boldsymbol{x} = \boldsymbol{e}$$
$$\boldsymbol{x_a} \in \{0,1\} \quad a \in A$$

where $\boldsymbol{x_a}$ denotes the variable associated with the a^{th} arc, and $\boldsymbol{M_a}$ its related column. We consider a basic integer solution $\bar{\boldsymbol{x}}$ to (P_3), and we denote by P the index set of all the positive-valued basic variables of $\bar{\boldsymbol{x}}$. The cocycle formulation allows the use of the compatibility defined by Zaghrouti et al. [21] for set partitioning problems.

Definition 2 *A nonempty set of columns D is said to be compatible with the basic integer solution $\bar{\boldsymbol{x}}$ (or simply compatible) if there is a subset of columns with indexes in $P^{'} \subseteq P$ such that $\sum_{a\in D} \boldsymbol{M_a} = \sum_{a\in P'} \boldsymbol{M_a}$. The set of variables corresponding to columns of D is also said to be compatible.*

The set of arcs composing a path π is denoted A^π, and the corresponding vector $\boldsymbol{x^\pi} \in \boldsymbol{\{0,1\}^{|A|}}$ is such that $x_a^\pi = 1$ *iff* arc $a \in A^\pi$. Recall that in the shortest path problem, a basic solution corresponds to a spanning tree with the source node as the origin. This solution consists of sending one unit of flow along the unique path π in the tree going from s to d. Consequently, the positive-valued variables correspond to the arcs $a \in A^\pi$, while the zero-valued variables correspond to the rest of the arcs. Furthermore, a compatible set of variables corresponds to a set of arcs in $A \setminus A^\pi$ that can replace (a subset of) A^π to form a new path. Using the notion of a cocycle introduced in Definition 1, we may view a set of compatible columns as a set of arcs that cover the same cocycles as those covered by a subset of the arcs in A^π. The next definition tightens the notion of compatibility using the minimality of a set of compatible columns.

Definition 3 *A set of arcs D is called a detour if it is compatible and minimal, in the sense that none of its strict subsets is compatible.*

Given a path π corresponding to a given basic solution, a detour may also be seen as a subpath with exactly one arc leaving π and one arc entering it, as illustrated in Figure 2.

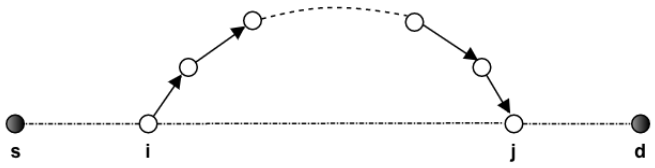

Figure 2: Notion of detour.

We now recall the definition of adjacency in linear programming.

Definition 4 *Let $\mathcal{P}$ be the polyhedron of a linear program. Two extreme points $\boldsymbol{x^1}$ and $\boldsymbol{x^2}$ of $\mathcal{P}$ are adjacent if there exists a face of $\mathcal{P}$ of dimension 1 (an edge) that contains both $\boldsymbol{x^1}$ and $\boldsymbol{x^2}$.*

The following proposition establishes the link between the notions of detour and path adjacency in $\mathcal{P}^{\mathcal{SPP}}$.

Proposition 1 *A path $\pi^{'}$ is adjacent to π iff there exists exactly one detour D such that $A^{\pi^{'}} \setminus A^\pi = D$.*

Proof. $\Leftarrow$ Consider a detour D such that $A^{\pi^{'}} \setminus A^\pi = D$, and let F be the face of least dimension containing x_π and $x_{\pi'}$. $F = \{\boldsymbol{x} : \boldsymbol{M}\boldsymbol{x} = \boldsymbol{e}\ ;\ x_a = 1\ \forall a \in A^\pi \cap A^{\pi^{'}}\ ;\ x_a = 0\ \forall a \in A \setminus (A^\pi \cup A^{\pi^{'}})\}$. On the one hand, $\boldsymbol{x_\pi}$

and $\boldsymbol{x_{\pi'}}$ are affinely independent, so $dim(F) \geq 1$ (1). On the other hand, if we denote by $F^{=}$ the matrix of the equality constraints satisfied by F, then $dim(F) = |A| - rank(F^{=})$.

Consider the subnetwork $G^*(V^*, A^*)$ composed of only the arcs in $(A^\pi \cup A^{\pi'}) \setminus (A^\pi \cap A^{\pi'})$. We have $|V^*| = |A^*| = |(A^\pi \cup A^{\pi'}) \setminus (A^\pi \cap A^{\pi'})|$. The number of cocycle equalities in G^* is then $|V^*| - 1 = |(A^\pi \cup A^{\pi'}) \setminus (A^\pi \cap A^{\pi'})| - 1$. As a result, $F^{=}$ is defined by at least $|A^\pi \cap A^{\pi'}| + |(A^\pi \cup A^{\pi'}) \setminus (A^\pi \cap A^{\pi'})| + |A \setminus (A^\pi \cup A^{\pi'})| - 1 = |A| - 1$ equalities that are linearly independent. Therefore, $rank(F^{=}) \geq |A| - 1$, which implies $dim(F) = |A| - rang(F^{=}) \leq |A| - |A| + 1 = 1$ (2). By (1) and (2), $dim(F) = 1$, so π_1 and π_2 are adjacent in $\mathcal{P}^{\mathcal{SPP}}$.

$\Rightarrow$ Suppose that π and π' are two adjacent paths, and there are $k > 1$ detours $D_1, D_2, ..., D_k$ such that $A^{\pi'} \setminus A^\pi = \cup_{i=1}^{k} D_i$. Let F be the face of degree 1 containing $\boldsymbol{x_\pi}$ and $\boldsymbol{x_{\pi'}}$. Consider the sequence of paths π_i such that $A^{\pi_i} \setminus A^\pi = D_i$, $i = \{1, 2, ..., k\}$. This implies $A^{\pi'} \setminus A^\pi = \cup_{i=1}^{k} (A^{\pi_i} \setminus A^\pi)$. Thus, $\boldsymbol{x_{\pi'}} - \boldsymbol{x_\pi} = \sum_i (\boldsymbol{x_{\pi_i}} - \boldsymbol{x_\pi})$, so $\boldsymbol{x_{\pi'}}$ is a linear combination of the set of $(k+1)$ extreme points $\{\boldsymbol{x_\pi}, \boldsymbol{x_{\pi_1}}, \boldsymbol{x_{\pi_2}}, ..., \boldsymbol{x_{\pi_k}}\}$. Therefore, these extreme points are contained in F. In addition, since they are affinely independent, we have $dim(F) \geq k > 1$. Hence, we have a contradiction. □

Let Adj_π be the set of all paths adjacent to π. The notion of adjacency can be extended to the general case where two extreme points are contained in a face of degree $k > 1$. We define the degree of adjacency as follows:

Definition 5 *In $\mathcal{P}^{\mathcal{SPP}}$, the degree of adjacency of two paths π and π' is equal to the smallest dimension k of a face containing $\boldsymbol{x^\pi}$ and $\boldsymbol{x^{\pi'}}$. We say that π and π' are k-adjacent.*

The next proposition gives a generalization of Proposition 1.

Proposition 2 *A path π' is k-adjacent to a path π iff there exist exactly k detours $D_1, D_2, ..., D_k$ such that $A^{\pi'} \setminus A^\pi = \cup_{i=1}^{k} D_i$.*

Proof. Let G^* be the subnetwork composed of only the arcs in $A^\pi \cup A^{\pi'}$.

$\Leftarrow$ Assume that there are k detours $D_1, D_2, ..., D_k$ such that $A^{\pi'} \setminus A^\pi = \cup_{i=1}^{k} D_i$. Let $\{\pi_1, \pi_2, ..., \pi_k\}$ be the set of all paths π_i adjacent to π in G^* such that $A^{\pi_i} \setminus A^\pi = D_i \ \forall i$, and let F be the smallest face containing x_π and x_{π_i} for $i = \{1, 2, ..., k\}$. On the one hand, the set $\{\boldsymbol{x_\pi}, \boldsymbol{x_{\pi_1}}, \boldsymbol{x_{\pi_2}}, ..., \boldsymbol{x_{\pi_k}}\}$ is affinely independent, so $dim(F) \geq k$. On the other hand, as in the previous proof, we can easily show that the rank of the equalities matrix $F^{=}$ is $\geq (|A| - 1)$, which leads us to conclude that $dim(F) = |A| - rank(F^{=}) \leq k$. Hence, $dim(F) = k$.

$\Rightarrow$ π and π' are k-adjacent means, so by Definition 5, k is the dimension of the smallest face F containing $\boldsymbol{x_\pi}$ and $\boldsymbol{x_{\pi'}}$. Suppose that $d < k$ is the number of detours between $\boldsymbol{x_\pi}$ and $\boldsymbol{x_{\pi'}}$. Then there exists a face of dimension d containing π and π', and F is no longer the smallest face. Suppose now that $d > k$. Since F contains π and π', it will contain the $(d+1)$ affinely independent extreme points $\{\boldsymbol{x_\pi}, \boldsymbol{x_{\pi_1}}, \boldsymbol{x_{\pi_2}}, ..., \boldsymbol{x_{\pi_d}}\}$ defined as follows: $\pi_1 \in Adj_\pi$, $\pi_i \in Adj_{\pi_{i-1}}$ for $i \in \{2, ..., d\}$ and $\boldsymbol{x_{\pi_d}} = \boldsymbol{x_{\pi'}}$. Thus, $dim(F) \geq d > k$. Hence, we have a contradiction. □

The last proposition can be interpreted as the existence of a sequence of k edges in $\mathcal{P}^{\mathcal{SPP}}$ linking $\boldsymbol{x_\pi}$ to $\boldsymbol{x_{\pi'}}$. In addition, the intermediate extreme points correspond to paths that are constructed of arcs and nodes of π and π'. The set of all paths k-adjacent to π is denoted Adj_π^k.

Corollary 3 *The sets of paths Adj_π^k for $k \in \{1, 2, ..., |A^\pi|\}$ form a disjoint partition of the set of all paths.*

Proof. Since the degree of adjacency is well defined (unique), the sets Adj_π^k are disjoint. Moreover, the partition size is upper bounded by the number of arcs $|A^\pi|$, since the maximum degree of adjacency is upper bounded by $|A^\pi|$. □

3.2 Adjacency to a set of paths

We now investigate the notion of adjacency of a path π to a set of paths Π. We denote by $A^\Pi = \cup_{\pi\in\Pi} A^\pi$ the set of arcs and by V^Π the set of nodes. We denote by $\mathcal{S}$ the set of all paths, while S^Π indicates the subset of $\mathcal{S}$ containing the paths composed of arcs of A^Π.

Definition 6 *Let P be the index set of all arcs in Π. A set of arcs D is said to be compatible with Π iff there are two subsets $P^+, P^- \subset P$ such that $\sum_{a\in D} \boldsymbol{M_a} = \sum_{a\in P^+} \boldsymbol{M_a} - \sum_{a\in P^-} \boldsymbol{M_a}$. If in addition D is minimal, we call it a detour.*

Remark: Similarly to the case of adjacency to a single path, a detour corresponds to a subpath containing exactly one arc leaving Π and one arc entering it, as shown in Figure 3.

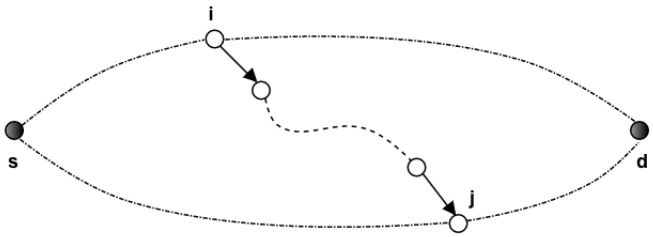

Figure 3: Detour from i to j for a set of two paths.

Definition 7 *A path π is k-adjacent to Π if there are exactly k detours $D_1, D_2, ..., D_k$ such that $A^\pi \setminus A^\Pi = \cup_{i=1}^k D_i$. We write $\pi \in Adj_\Pi^k$.*

Proposition 3 *If a path π is k-adjacent to Π, then there is at least one path π_0 in S^Π and $1 \leq t \leq k$ such that $\pi \in Adj_{\pi_0}^t$.*

Proof. $\pi \in Adj_\Pi^k$ implies the existence of k detours $D_1, D_2, ..., D_k$ such that $A^\pi \setminus A^\Pi = \cup_{i=1}^k D_i$. Let π_0 be the path of S^Π that shares the maximum number of arcs with π. The number of detours made by π in relation to π_0 is upper bounded by k, so $\pi \in Adj_{\pi_0}^t$ while $t \leq k$. Moreover, it is obvious that $t \geq 1$ since $k \geq 1$. □

Consider a path π that is $k-$adjacent to a set of paths Π. We show below that there is an affine basis generating S^Π such that π is at most $(k+1)-$adjacent to every path in the basis. We need the following preliminary result:

Proposition 4 *Consider a path π_0 and a sequence of sets Π_k such that $\Pi_0 = \{\pi_0\}$ and $\Pi_i = \Pi_{i-1} \cup \{\pi_i\}$ where $\pi_i \in Adj_{\Pi_{i-1}}^1$, $i \in \{1, 2, ..., |A| - |V| + 1\}$. The set $B = \{\boldsymbol{x}_{\pi_i}, i = 0, 1, ..., |A| - |V| + 1\}$ is an affine basis of $\mathcal{P}^{\mathcal{SPP}}$.*

The next lemma is useful for proving this result.

Lemma 1 *The dimension of $\mathcal{P}^{\mathcal{SPP}}$ is equal to the number of detours needed to cover the whole network.*

Proof. The proof is by induction on the dimension of the polyhedron. For a 0-dimensional polyhedron, there is exactly one path in the network and 0 detours, and the lemma holds. Let $G^n(V^n, A^n)$ be an acyclic network and $\mathcal{P}^n$ the polyhedron of paths in G^n. By Corollary 2, we have $dim(\mathcal{P}^n) = |A| - |V| + 1$. Suppose that $dim(\mathcal{P}^n) = n$ where n is the number of detours needed to cover G^n. We add to the network one detour D with a number of arcs $|D|$. The number of newly added nodes is then equal to $|D| - 1$. We call the resulting network $G^{n+1}(V^{n+1}, A^{n+1})$, and its corresponding polyhedron is $\mathcal{P}^{n+1}$. There are $|V^{n+1}| - 1 = |V| + |D| - 1$ linearly independent cocycle equality constraints. Thus, $dim(\mathcal{P}^{n+1}) = |A^{n+1}| - |V^{n+1}| + 1 = |A| + |D| - (|V| + |D| - 1) + 1 = |A| - |V| + 2 = dim(\mathcal{P}^n) + 1 = n + 1$. This completes the proof. □

Proof of Proposition 4. Consider a path π_0 and a sequence of sets Π_k such that $\Pi_0 = \{\pi_0\}$ and $\Pi_i = \Pi_{i-1} \cup \{\pi_i\}$ where $\pi_i \in Adj^1_{\Pi_{i-1}}$ $i \geq 1$. By Lemma 1, there are $|A| - |V| + 1$ detours needed to cover the whole network, where each detour generates a path π_i that is affinely independent of the set of paths Π_{i-1}. This gives a total of $|A| - |V| + 1$ affinely independent paths. When we add the initial path π_0, the set $\{\boldsymbol{x}_{\pi_i}, i = 0, 1, ..., |A| - |V| + 1\}$ is affinely independent and generates $\mathcal{S}$, so it forms an affine basis of $\mathcal{P}^{\mathcal{SPP}}$. □

Proposition 5 *Let π be a k-adjacent path to Π. There is an affine basis $B = \{\boldsymbol{x}_{\boldsymbol{\pi_i}}\}_{(i=0,1,...,n)}$ generating S^Π such that $\forall i \in \{0, 1, ..., n\}$, $\exists$ t such that $1 \leq t \leq k + 1$, $\pi \in Adj^t_{\pi_i}$.*

Proof. Consider $\pi \in Adj^k_\Pi$. There is at least one path in $S^\Pi \cap Adj^k_\pi$; let π_0 be one of these paths. We choose a series of paths $\pi_i \in S^\Pi$ such that $\pi_i \in Adj^1_{\Pi_{i-1}} \cap Adj^1_{\pi_0}$, $\Pi_0 = \{\pi_0\}$, $\Pi_i = \Pi_{i-1} \cup \{\pi_i\}$, and $\Pi_n = \Pi$. Since $\pi \in Adj^k_{\pi_0}$, π is at most $(k + 1)$-adjacent to π_i $\forall i$. Moreover, from Proposition 4 the set $\{\boldsymbol{x}_{\boldsymbol{\pi_0}}, \boldsymbol{x}_{\boldsymbol{\pi_1}}, ..., \boldsymbol{x}_{\boldsymbol{\pi_n}}\}$ is affinely independent and generates S^Π. B is then an affine basis generating S^Π. □

Remark 1 *Let Π be a set of paths and B the affine basis generating S^Π. We note that every path in S^Π that is a combination of the elements of the basis B is 0-adjacent to Π. These paths will be generated by the Combination step of the PAB algorithm that will be introduced in Section 4.*

Proposition 6 *Every set of paths defines a face of $\mathcal{P}^{\mathcal{SPP}}$.*

Proof. Let $G^\Pi(A^\Pi, V^\Pi)$ be a subnetwork of a network $G(V, A)$. The dimension of the polyhedron corresponding to subnetwork G^Π is $|A^\Pi| - |V^\Pi| + 1$. According to Proposition 4, we can extract exactly $|A^\Pi| - |V^\Pi| + 2$ affinely independent paths from G^Π. Consequently, this set of paths generates a face of $\mathcal{P}^{\mathcal{SPP}}$, with dimension $|A^\Pi| - |V^\Pi| + 1$. □

Proposition 7 *Every path is an affine combination of the elements of the basis B.*

Proof. Let π be a path in $G(V, A)$. $B = \{\boldsymbol{x}_{\boldsymbol{\pi_i}}, \boldsymbol{i = 0, 1, ..., |A| - |V| + 1}\}$ is a basis. Thus, there exists α_i such that $x_\pi = \sum_i \alpha_i \boldsymbol{x}_{\boldsymbol{\pi_i}}$. Moreover, we know that every path satisfies the cocycle constraints. Thus, $\boldsymbol{M x_\pi} = \sum_i \alpha_i \boldsymbol{M x_{\pi_i}} = (\sum_i \alpha_i)\boldsymbol{e} = \boldsymbol{e}$, which implies that $\sum_i \alpha_i = 1$. □

Proposition 8 *The sets of paths Adj^k_Π for $k \in \{1, 2, ..., k_{max}\}$ form a disjoint partition of $\mathcal{S}$, where k_{max} is an upper bound on the degree of adjacency to Π.*

Proof. Every path $\pi \in S$ has a unique degree of adjacency in relation to a set of paths Π. The partition $S = \cup_k Adj^k_\Pi$ is then a disjoint partition of the set of paths. □

4 Primal adjacency-based algorithm

Globally, PAB is the result of the combination of DP and the polyhedral concepts and results introduced in the previous section. The main features of our PAB algorithm are the following: first, instead of executing one search in the whole network, the algorithm performs a DP sequence in limited search spaces. This search space reduction allows the algorithm to explore larger networks. Second, the algorithm can use the specific structure of the application networks to construct an initial set of paths rich in primal information. This set is dynamically improved at each iteration until optimality is reached. Third, thanks to the combination property introduced in Proposition 7, the algorithm can combine the generated paths to produce good improving paths in a limited computational time.

We introduce the PAB pseudocode and describe its main steps in Section 4.1. We then present in Section 4.2 an improved labeling algorithm (ILA) that is the main component of the PAB algorithm. Finally, we discuss its convergence in Section 4.3. We introduce the following notation.

k	Iteration/degree of adjacency.
k_{max}	Maximum degree of adjacency.
Z^*	Current best SPPRC cost.
Π	Current set of generated paths.
C_{di}	Reverse SPP cost from $i \in V$ to the destination d.
U_i	Cost upper bound for node $i \in V$.
$\mathcal{L}_i$	Active labels in node i; $\mathcal{L} = \cup\mathcal{L}_i$.
$\mathcal{S}_i^k$	Saved labels of degree k in node i; $\mathcal{S}_i = \cup\mathcal{S}_i^k$; $\mathcal{S} = \cup\mathcal{S}_i$.
Ω	Set of Pareto-optimal paths provided by ILA.

4.1 PAB pseudocode

Starting from an initial set of paths $\Pi = \Pi_0$, the algorithm iteratively explores a partition of the feasible domain using the degree of adjacency as a measure of the distance to Π_0. This measure provides a disjoint partition of the solution space, as mentioned in Proposition 8. The search proceeds from lower degrees of adjacency to higher degrees. At the end of each iteration the promising paths found are added to Π. These paths are used for two purposes. First, they are combined in the Combination step to generate improving paths with higher degrees of adjacency; second, their best cost helps to tighten the cost upper bounds on every node, allowing the elimination of many useless labels. This drastically reduces the computational complexity of the subsequent iterations. Below we give the PAB pseudocode and describe its four main steps.

Algorithm 1 Primal adjacency-based algorithm for the SPPRC

Initialization step
Find an initial set of paths Π_0 and compute $C_{di}, i \in V$.
$k \leftarrow 1$; $\mathcal{S}_s \leftarrow \{(0, ..., 0)\}$, $\mathcal{S}_i \leftarrow \emptyset, \forall i \neq s$; $\Pi \leftarrow \Pi_0$.
Combination step
$\mathcal{L}_i \leftarrow \mathcal{S}_i$ for all $i \in V^{\Pi}$.
$\Omega \leftarrow ILA(k_{max}, G^{\Pi}, \mathcal{L}, \mathcal{S})$.
Extension step
$\mathcal{L}_i \leftarrow \mathcal{S}_i^k$ for all $i \in V$.
$\Omega \leftarrow ILA(k, G, \mathcal{L}, \mathcal{S})$.
Control step
if $k = k_{max}$ **then**
  Stop: the solution is optimal; return Π.
$k \leftarrow k + 1$.
if $\Omega \neq \emptyset$ **then**
  $\Pi \leftarrow \Pi \cup \Omega$; go to Step 1.
else
  go to Step 2.

As mentioned earlier, a subpath π_i in node i is characterized by a label $l_i = (C_{l_i}, R^1_{l_i}, R^2_{l_i}, ..., R^{|R|}_{l_i})$, which is an (—R—+1)-vector with values representing the cost and resource consumption up to node i. A new *adjacency* resource $R^{adj}_{l_i}$ is added to the label definition to count the degree of adjacency to the path set Π_0. This resource counts the number of detours needed to construct π; its upper bound is set to k for all the nodes. Based on Definition 7, it simply counts the number of arcs in path π that leave the set Π_0, and hence the corresponding arc consumption is 1 on the arcs leaving set Π_0 and 0 otherwise. Note that the new adjacency resource is used only to limit the search space; it does not contribute to the dominance rules.

Initialization step. In this step, we initialize Π with a path set Π_0 that contains some good primal information. For crew scheduling problems, primal information is generally available: it can be extracted from previous schedules or taken from the predetermined bus lines or aircraft routes. We also compute $C_{di}, i \in V$ (by finding a reverse shortest path from d to s) for use in the Combination and Extension steps to tighten the upper bounds on the costs for all the nodes of the network. This preprocessing technique has also been embedded in the standard DP algorithm, so that we can fairly compare the two approaches.

Combination step. Given a set of paths Π, this step seeks improving paths that are affine combinations of paths composing Π, as described in Proposition 7. Technically speaking, this step consists of running ILA on G^{Π}, with a degree of adjacency to Π_0 equal to k_{max}, which means that the adjacency resource is not active. From a polyhedral point of view, this step may be viewed as a 0-*adjacency* search on the smallest face F

containing the extreme points corresponding to the paths of Π. It searches for extreme points contained in F whose corresponding paths have not been generated yet. These paths generally have a higher degree of adjacency to Π_0 compared to the previously found paths that compose Π.

Moreover, since this step is performed in a restricted subgraph G^{Π} of G, the set of labels corresponding to subpaths leaving G^{Π} are saved in $\mathcal{S}$ in order to be extended in the subsequent iteration corresponding to their degree of adjacency. In practice, this step is relatively fast, and it achieves significant cost improvements, as our experimental results will demonstrate. It is efficient because it considers relatively small subgraphs with primal information and because it generates improving paths of higher degrees of adjacency even before the PAB algorithm reaches these degrees.

Extension step. This step extends the search space by increasing the degree of adjacency. We search here for improving paths having a higher adjacency degree when no such solutions are found in lower degrees. The search is performed by running ILA on the original network G, with a degree of adjacency k and a set of saved labels $\mathcal{S}$ from the previous iteration. Note that all the newly saved labels are of degree $k+1$. An updated set $\mathcal{S}$ of saved labels to be used in the subsequent iteration is produced.

Control step. If the maximum degree is reached, we assert that the current solution is optimal and stop the algorithm. Otherwise, if improving paths are found for degree of adjacency k, they are added to the set of paths Π. This is done by adding to A^{Π} all the arcs for the detours of the paths. In this case, a new Combination step is carried out in the updated set Π. Otherwise, the algorithm enlarges the search space to the next degree of adjacency and performs a new search in the corresponding neighborhood.

From a polyhedral point of view, the PAB algorithm looks for a better solution among the extreme points with a given degree of adjacency in relation to Π_0, and it increases this degree. By Proposition 6, these sets of paths define a face F_0 in $\mathcal{P}^{\mathcal{SPP}}$. PAB enlarges the basis generating F_0 at each iteration by adding to it a set of newly generated paths, which creates a new face of larger dimension containing F_0. Finally, note that PAB does not need to find all the $|A|-|V|+1$ elements of the basis generating S to prove optimality; it adds only the most promising ones. The reason is that PAB is able to produce paths having different degrees of adjacency to Π_0 (≥ 1) and not only those of degree 1. These k-adjacent paths contain the most promising detours that may be used to produce improving paths using affine combinations. We recall that a path π is k-adjacent to a set of paths Π if there exists a basis B generating Π such that every path in B is at most $(k+1)$-adjacent to π (Proposition 4).

4.2 Improved Labeling Algorithm

Recall that a labeling algorithm for SPPRC is an extension of the classical Bellman algorithm for SPP [15]. The algorithm starts from a trivial subpath containing only the source node s and iteratively constructs new paths by extending available subpaths one-by-one in every direction satisfying the resource constraints imposed on the nodes. ILA is an improved labeling algorithm. It allows the generation of a set of feasible paths for a given degree of adjacency in an acyclic network $G(V,A)$ that is assumed to be topologically ordered. Given an adjacency degree k, a working graph $\bar{G}(\bar{V},\bar{A})$, a set of active labels $\mathcal{L}_i$, and a set of saved labels $\mathcal{S}=\cup_i\mathcal{S}_i$, $ILA(k,\bar{G},\mathcal{L},\mathcal{S})$ returns a set of Pareto-optimal paths Ω that are at most k degrees distant from Π_0. It also saves in $\mathcal{S}$ all the labels for efficient subpaths with higher degrees, or those that cannot be extended in $\bar{G}$, to avoid regenerating them in the subsequent iterations. The ILA pseudocode (Algorithm 2) is given below.

Algorithm 2 Improved Labeling Algorithm $ILA(k, \bar{G}, \mathcal{L}, \mathcal{S})$

1: **for all** $i \in V$ **do**
2: Update cost upper bound: $U_i \leftarrow Z^* - C_{di}$.
3: **for all** $i \in \bar{V} \setminus \{d\}$ such that $\mathcal{L}_i \neq \emptyset$ (in increasing order) **do**
4: Delete dominated labels from $\mathcal{L}_i$.
5: **for all** $j \in V$ successor of i **do**
6: **for all** $l_i \in \mathcal{L}_i$ **do**
7: $l_j \leftarrow Extend(l_i, j)$.
8: **if** $j \in \bar{V}$ and l_j is feasible **then**
9: $\mathcal{L}_j \leftarrow \mathcal{L}_j \cup \{l_j\}$.
10: **else**
11: **if** $j \notin \bar{V}$ or $k < R^{adj}_{l_j} \leq k_{max}$ (the only resource violated) **then**
12: $\mathcal{S}_j \leftarrow \mathcal{S}_j \cup \{l_j\}$.
13: Build Ω from $\mathcal{L}_d$ and update Z^*.
14: **Return** Ω.

In Step 2, we tighten the cost upper bound on the nodes to fathom the unpromising labels. The new upper bound at node i is obtained by subtracting C_{di}, the cost of the shortest path from i to the destination node d, from the best current cost Z^*. Therefore, a sequence of nested sets of paths of nonincreasing cost is generated during the PAB process. This helps to further reduce the number of labels generated at each iteration, already reduced by the neighborhood partition w.r.t. the degree of adjacency.

In Step 4, we remove dominated labels. A label $l = (C_l, R^1_l, R^2_l, ..., R^{|R|}_l, R^{adj}_l)$ dominates $l' = (C_{l'}, R^1_{l'}, R^2_{l'}, ..., R^{|R|}_{l'}, R^{adj}_{l'})$ if $C_l \leq C_{l'}$ and for all components $t = 1, ..., |R|$, the inequality $R^t_l \leq R^t_{l'}$ holds. Observe that we do not dominate on the adjacency resource. The efficiency of a labeling algorithm is a result of its ability to identify and discard subpaths that cannot contribute to an optimal path. It discards both infeasible subpaths and subpaths that may not lead to a better (feasible) solution than a given subpath. For more details on labeling algorithms and dominance principles and rules see [15].

In Step 7, we call the $Extend(l_i, j)$ function, which extends a label l_i to a node j using predetermined extension rules. The most commonly used rules are as follows: $C_{l_j} := C_{l_i} + c_{ij}$, $R^t_{l_j} := \max\{a^t_i, R^t_{l_i} + r^t_{ij}\} \forall k \in 1, ..., |R|$, $R^{adj}_{l_j} := R^{adj}_{l_i} + r^{adj}_{ij}$; other rules may appear in real applications.

If the extended label l_j is feasible in terms of the SPPRC constraints and respects the current adjacency degree, it is added in Step 9 to the set of active labels in node j. Otherwise, if it violates the adjacency resource constraints while respecting the original SPPRC constraints, it is saved in node j to be used in the subsequent iteration(s). It is also saved if node j is not contained in the working graph $\bar{G}$ (mainly in the Combination step). We note that a label of degree k in node j is saved in $\mathcal{S}^k_j$. In Step 12, we use $\mathcal{S}_j$ instead of $\mathcal{S}^k_j$ to simplify the notation. The infeasible labels in terms of the SPPRC constraints are obviously ignored, since they are not added to $\mathcal{L}_j$ or $\mathcal{S}_j$. The motivation is to allow the PAB algorithm to continue the extension of the labels that have been previously generated, instead of regenerating them from scratch at each iteration.

In Steps 13–14, we build Ω from Pareto-optimal labels of the destination node, we recompute Z^* via $\min\{Z^*, C_\pi : \pi \in \Omega\}$, and we return Ω.

4.3 Convergence analysis

We now discuss some important features of the PAB algorithm. We first prove that no label is produced more than once by the algorithm. We then show that efficient labels in DP are also efficient for PAB, before concluding that the proposed algorithm terminates by finding the optimal solution, given a predetermined maximum degree of adjacency.

Proposition 9 *No label is generated more than once by the PAB algorithm.*

Proof. Consider a label l of degree of adjacency k generated in iteration k. The labels saved in the Extension step in iteration k are all of degree of adjacency $k + 1$. Some of these labels that are saved in nodes of G^π

are extended in the next Combination step, while others are extended in the next Extension step associated with higher degrees of adjacency. The newly created labels are all of degree $k + t$ ($t \geq 1$). Consequently, label l will never be regenerated in future iterations. □

Proposition 10 *If a label l_i is efficient (Pareto optimal) in node i for the DP algorithm, it is also efficient in i in one of the iterations of the PAB algorithm, unless it is eliminated by cost bounding.*

Proof. Let l_i be an efficient label in node i for DP. This implies that there is no label from the source node s to node i of all degrees that dominates l_i. In particular, given a fixed degree k, there is no label of degree k that dominates l_i in node i. Label l_i is then efficient in PAB as well unless it is eliminated by cost bounding. This is true provided that all degrees of adjacency are covered by PAB. □

The next proposition shows that the optimal value of the SPPRC cannot increase from one iteration to the next or within the same iteration between two calls to ILA.

Proposition 11 *For any two consecutive calls to ILA, the cost strictly decreases.*

Proof. At the end of each call to ILA, whether in a Combination step or in an Extension step, the best cost found Z^* is used to tighten the cost upper bound for each node in the network. Thus, for a given node i, the cost C_l of each feasible label $l \in \mathcal{L}_i$ satisfies $C_l < U_i = Z^* - C_{di}$. In particular, in the destination node d where $C_{dd} = 0$, we have $C_l < Z^*$. The cost is thus strictly decreasing until optimality. □

Proposition 12 *The PAB algorithm terminates by finding the optimal solution.*

Proof. Based on Proposition 10, all efficient labels in DP are efficient for a certain degree of adjacency in PAB. In particular, the efficient labels in the destination node d in DP are efficient in PAB unless they are eliminated by cost bounding, which is not the case for the optimal solution. The exactness of the algorithm is thus guaranteed once k_{max} is achieved. □

Note that as a primal method, PAB returns a set of primal feasible solutions at the end of each iteration. This feature, which is not offered by classical DP methods, has two important advantages. First, the best cost returned at the end of each iteration is used to tighten the cost bounds, which greatly reduces the combinatorial complexity. Second, the primal aspect of PAB is a good match with the requirements of the CG method. The CG subproblems seek negative reduced cost paths, and these paths can be obtained using PAB in reasonable computational times.

5 Experimental results

To evaluate the efficiency of PAB, we conducted a series of computational experiments. The tests were performed on instances of the simultaneous vehicle and crew scheduling problem (VCSP) used by Haase et al. [12] in their study of urban transit systems. This problem is representative of a broad class of general VCSPs both in terms of the size of the networks and the number of resources. In some airline instances, we have to deal with tens of resources, although dominance is in practice performed on only 4 to 5. For the tests, we dominate on 7 resources, a number that is large enough to simulate the most complex situations in airline transportation. Furthermore, good initial points can easily be obtained for these problems, so they provide a good framework for evaluating the efficiency of primal solution methods. We will give a brief definition of the VCSP, followed by some implementation tips, before presenting the computational results.

5.1 VCSP instances

5.1.1 VCSP overview

Given a predetermined set of bus lines in a city and a set of timetabled trips to operate on these lines, the VCSP simultaneously determines bus and crew schedules that cover the set of trips at a minimum cost. A timetabled trip is divided into several consecutive segments, called *d-trips*. At locations called *relief points* drivers can change bus lines or return home or to the depot. Empty moves called *deadheads* are used to reposition buses. The set of consecutive d-trips and deadheads performed by a driver is called a *piece of work*, and two pieces of work are separated by a *break*. A work day for a driver, called a *duty*, is a sequence of pieces of work and breaks.

Numerous formulations have been proposed for the VCSP. Most of them formulate the problem via a set partitioning model and solve it using branch and price. The constraints that assign exactly one driver to each d-trip form the core of the master problem. We note that the bus schedules are usually derived a posteriori from the driver schedules. To facilitate this, supplementary constraints are usually added to the master problem. In addition to these constraints, the driver schedules are restricted by a variety of rules defined by the collective agreements and internal regulations. These rules are modeled in the subproblems using resource constraints. Each subproblem is an SPPRC on a space-time network, and it is solved to generate feasible driver schedules. A schedule is said to be feasible if it satisfies seven resource constraints: the minimum and maximum number of pieces of work; the maximum time of a duty, of a break, and of work in a duty; and the minimum and maximum length of a piece of work. For more details about the VCSP, see [12].

The complexity of the SPPRC is a result of the number of resources and the length of the resource intervals. Table 1 is reproduced from [12]; it gives the lower and upper bounds of the resource intervals as specified by the work rules. These intervals and the reduced cost interval are wide, which makes the problem difficult: we may have thousands of nondominated labels per node.

Table 1: Work rules for a driver schedule.

	Minimum	Maximum
No. of pieces	1	3 or 4
Piece length (mins)	15	300
Duty length (mins)	45	600
Work time (mins)	30	480
Break time (mins)	15	90

5.1.2 Instance generation and initial point

The VCSP instances differ in 3 parameters: the number of trips to be covered (120, 160, 200, or 240); the number of relief points per trip (5, 7, or 9); and the number of pieces of work per duty (3 or 4). We created a total of 24 instance types: 12 with 3 pieces of work and 12 with 4 pieces. For each type, we randomly generated 5 instances using the generator of [12]. This gives a total of 120 VCSP instances with up to 1 million arcs (see Table 2).

The VCSP is usually solved by branch and price. We used CG to solve the LP relaxation of these VCSP instances. For each VCSP instance, we captured randomly two pricing subproblems, one from the first iterations of CG process and the second one from the last iterations. These 240 pricing subproblems form our test instances. This allows for a fair comparison between PAB and DP (same well-defined instances with well-defined reduced costs). We do not report results of the integration of PAB into the CG process to avoid side effects from the LP solver used for the master problem (mainly its impact on the dual variables used to compute the reduced costs) and other CG strategies used to accelerate convergence.

We use “3-pieces-b”, “3-pieces-e” to denote respectively instances with 3 pieces of work taken form the beginning of CG and from the end. Similarly, “4-pieces-b”, “4-pieces-e” denote instances with 4 pieces of work. We also use “b-instances” and “e-instances” to distinguish instances extracted form the beginning of CG from those extracted from the end.

Table 2: List of test instances.

Instance types	# trips	# relief points	# d-trips	# arcs	# nodes
5_120	120	5	720	78989.2	50690.0
5_160	160	5	960	141286.0	91458.4
5_200	200	5	1200	220006.8	143100.4
5_240	240	5	1440	314692.6	205377.4
7_120	120	7	960	152192.4	98891.2
7_160	160	7	1280	273755.4	178959.6
7_200	200	7	1600	427856.4	280649.0
7_240	240	7	1920	612310.2	402557.6
9_120	120	9	1200	249093.6	162862.0
9_160	160	9	1600	450255.6	295783.2
9_200	200	9	2000	703604.8	463421.0
9_240	240	9	2400	1008197.8	665201.0

We constructed an initial point (see Section 5.1) by assigning to each timetabled trip a path that starts from the depot, covers the trip, and returns to the depot. These initial paths are added to $\Pi = \Pi_0$, and their arcs are added to A^{Π}. These paths may be infeasible, but they contain a good percentage of the primal information.

We choose this initial point for the following reasons. First, in crew scheduling problems, the crews do not often change vehicles, so the crew schedules have many pieces in common with the bus lines. Second, in a reoptimization context, we observe that the reoptimized paths are characterized by a slight deviation from the planned paths: a high percentage of the arcs and nodes of the initial point remain in the final reoptimized Pareto-optimal paths. Third, the construction of the initial point does not require additional computational time since all the information is available.

5.1.3 Theoretical results related to VCSP instances

We know from Definition 7 that a path π is k-adjacent to a set of paths Π_0 if k detours are needed to construct the path π using Π_0. Based on the initial point above, each detour corresponds to a break between two consecutive pieces of work in a schedule. The maximum number of allowed detours k_{max} is then the maximum number of daily pieces of work minus one. Hence, the adjacency resource corresponds to a resource that counts the number of pieces of work. Under this condition, we can prove that the number of labels generated by all Extension steps in G is less than or equal to the number generated by DP. If we consider in addition the effect of cost bounding, the inequality becomes strict. Thus, PAB creates fewer labels than DP.

Proposition 13 *The number of labels created by the PAB algorithm in all Extension steps is less than or equal to the number created by the DP algorithm.*

Proof. First, by Proposition 9, no label is generated more than once by PAB. Second, the feasibility conditions in terms of resource constraints are the same for the two algorithms. Third, suppose that there is a label l_i of degree k that is dominated at node i by DP but not by PAB. Let $l_i^{'}$ be the efficient label that dominates l_i. There are three cases. Case 1: If the degree of $l_i^{'}$ is strictly greater than k, dominance is not possible, because the degree of adjacency corresponds to the number of detours, which is a counter corresponding to one of the original resources of the problem (the number of pieces of work). Case 2: $l_i^{'}$ is of degree k. In this case, $l_i^{'}$ dominates l_i at the k^{th} iteration of PAB as well. Case 3: The degree of $l_i^{'}$ is strictly less than k. in this case, $l_i^{'}$ is generated in the previous iterations and saved to be used to dominate l_i in the subsequent iterations. Therefore, every label eliminated by dominance in DP is similarly eliminated in PAB. Moreover, many labels can be eliminated at each iteration of PAB due to cost bounding, which is not possible for DP; this completes the proof. □

It is possible that some of the labels created in the Combination step are not generated by DP. This is because the Combination step may generate labels of degree $k+t$ where $t \geq 1$, while only degree k is achieved by the Extension step. This implies that labels of degree $\geq k+t$ that may dominate those generated in the

Combination step are not yet available for PAB. However, since the Combination steps are performed in very small subgraphs G^{Π}, this side effect is insignificant, as our experiments will show.

Proposition 14 *The number of calls to dominance function in all Extension steps is strictly less than the number of calls to dominance function by DP.*

Proof. Let $n = \sum_{k=0}^{k_{max}} n_k$ be the number of feasible labels in a given node of G, where n_k is the number of labels with a degree of adjacency k regarding the initial point Π_0. The number of calls to dominance function by DP is in the worst case C_n^2. This number is equal to $C_{n_k}^2$ for the Extension step related to an iteration k of PAB, so the total number of calls in all Extension steps is $\sum_{k=0}^{k_{max}} C_{n_k}^2$ which is largely smaller than C_n^2. In addition, by Proposition 13, the total number of labels generated by PAB in each node is at most equal to the number of labels generated by DP; this completes the proof. □

Finally, the notion of a detour is not specific to the VCSP. In airline crew scheduling, we observe that pilots and copilots do not often change planes; their rotations (pairings) thus have many pieces in common with the predetermined plane routes, and are generally similar to the schedules of previous months. If we consider these routes or previous schedules as the initial point, every deviation can be viewed as a detour. Some commercial solvers speed up the solution of these problems by limiting the generation of pairings to one detour per duty for the first 90% of the CG iterations. In these cases the number of detours in a path is two or three, corresponding to the mean number of duties in a pairing. In the final 10% of the iterations more detours are permitted, but the subproblems remain easy because there are many paths with reduced costs close to zero, and dominance greatly reduces the number of Pareto optimal paths to consider. Even if there is no resource counting the number of pieces of work in airline crew scheduling problems, the basic notion of a detour is implicitly present. Our work is potentially adaptable to many transportation problems.

5.2 Computational results

The instances described above have been solved by both DP and PAB on a 2.5 Ghz Intel Core i5 MacBook. The DP solver used is the label setting algorithm of BOOST, a well-known C++ library. Henceforth, we refer to it as std. DP. The rest of this section compares PAB with MDDPA and std. DP, which is the standard technique used by commercial solvers for the SPPRC. We note that MDDPA [14] provides two search strategies, Nearest First and Best First. We use the former because it is more efficient.

5.2.1 Computational time

PAB performs well in terms of computational time: first, it is considerably faster than std. DP; second, it is able to find optimal solutions earlier than MDDPA does for all the test instances; third, it is better than MDDPA at proving optimality for b-instances while being competitive for e-instances. Tables 3 and 4 compare the computational times of the three methods. The CPU column gives the computational time, and the Opt. time column gives the time required by PAB and MDDPA to find an optimal solution before proving its optimality. As we can see, Opt. time is a fraction of CPU.

We compute the time reduction factors realized by PAB as the ratio of the std. DP time to the PAB time. The average ratio is about 2.53 for the b-instances and 3.21 for the e-instances. The largest reduction factors occurred for the largest instances. PAB is more efficient than MDDPA for the b-instances, since the reduction factors of the latter do not exceed 1.97 with an average of 1.89. However, MDDPA is better than PAB for the e-instances, with an average reduction factor of 4.21. Figure 4 shows the PAB time reduction factors for different classes of instances.

Tables 3 and 4 also give the time when an optimal solution is found by PAB and MDDPA. PAB performs better for all the classes of instances. It returned optimal solutions in an average of 24.75% of the total std. DP time for 3-piece-b instances and about 16.42% for 4-piece-b instances. The corresponding figures for MDDPA were 39.01% and 29.74%. PAB returned optimal solutions in an average of 11.36% for 3-piece-e instances and about 7.41% for 4-piece-e instances; the corresponding figures for MDDPA were 17.89% and

12.27%. Since the complexity of the SPPRC grows significantly with the size of the network and the number of pieces of work per duty, we conclude that the efficiency of PAB is more significant for more complex instances.

Table 3: Computational times for 3-piece-b and 3-piece-e instances.

	b-instances					e-instances				
	std. DP	MDDPA		PAB		std. DP	MDDPA		PAB	
Inst. type	CPU	CPU	Opt. time	CPU	Opt. time	CPU	CPU	Opt. time	CPU	Opt. time
5_120	3.56	2.21	1.05	1.71	1.08	1.18	0.17	0.17	0.37	0.09
5_160	9.64	5.25	2.58	3.57	2.02	3.37	1.38	0.49	1.39	0.32
5_200	21.53	10.93	6.67	8.04	6.05	6.97	4.05	1.26	3.12	0.76
5_240	33.65	21.01	16.42	14.51	6.27	12.17	3.91	2.34	3.62	0.98
7_120	8.84	4.76	3.22	3.84	2.99	2.32	0.34	0.34	0.64	0.17
7_160	18.93	15.30	10.63	9.29	5.00	8.77	2.36	0.95	2.86	0.72
7_200	47.10	30.13	25.44	20.68	12.02	18.17	6.14	3.45	5.70	2.74
7_240	95.55	56.40	34.70	31.38	12.98	26.89	8.98	6.53	7.62	3.83
9_120	15.29	9.71	5.97	6.94	6.20	5.69	1.48	0.69	1.71	0.84
9_160	41.98	24.95	14.99	14.74	10.93	16.19	7.10	4.13	6.07	2.12
9_200	95.95	55.06	37.77	35.53	20.13	42.38	12.83	4.41	12.19	6.40
9_240	176.64	114.91	61.69	56.73	21.17	56.54	18.20	17.76	12.28	7.03

Table 4: Computational times for 4-piece-b and 4-piece-e instances.

	b-instances					e-instances				
	std. DP	MDDPA		PAB		std. DP	MDDPA		PAB	
Inst. type	CPU	CPU	Opt. time	CPU	Opt. time	CPU	CPU	Opt. time	CPU	Opt. time
5_120	6.81	2.88	1.58	2.95	1.69	1.79	0.18	0.18	0.52	0.10
5_160	18.34	7.83	3.66	7.14	2.25	5.08	1.58	0.41	2.37	0.32
5_200	46.86	18.74	12.20	15.95	6.51	10.43	5.47	1.28	6.27	0.77
5_240	65.67	35.69	30.55	28.69	11.09	18.61	4.67	2.36	6.08	1.42
7_120	14.02	6.62	4.83	6.42	3.23	3.58	0.37	0.37	0.92	0.16
7_160	33.00	20.03	11.10	16.31	4.21	12.57	2.79	0.99	4.76	0.94
7_200	111.08	61.87	31.15	50.01	15.34	28.02	7.35	4.06	10.35	2.72
7_240	194.96	92.26	49.88	66.06	27.71	43.36	11.36	6.72	12.47	3.74
9_120	29.06	13.78	8.17	12.31	5.69	9.97	1.85	0.77	2.23	0.83
9_160	89.46	39.21	22.68	29.00	10.46	28.90	10.76	5.85	11.23	2.17
9_200	252.62	103.70	65.58	81.07	25.26	72.92	17.17	4.08	22.27	6.39
9_240	343.79	189.01	137.69	152.13	82.72	97.30	22.73	22.27	19.50	7.08

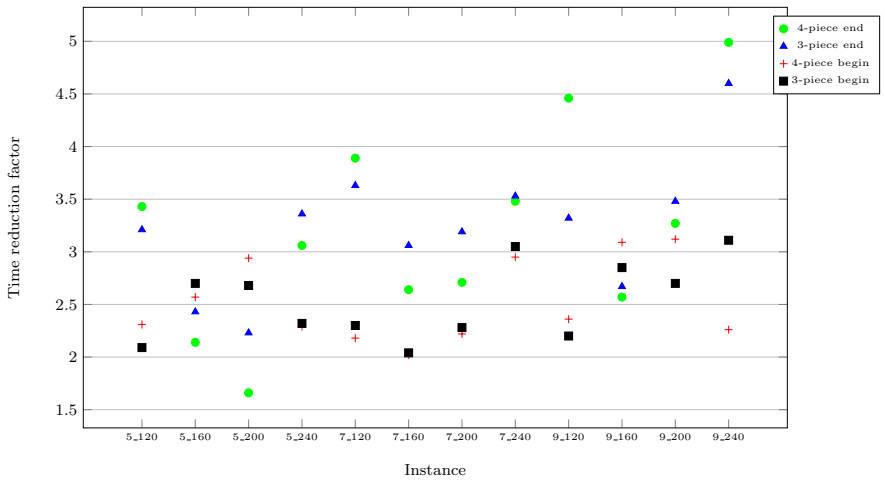

Figure 4: Time reduction factor (PAB vs. std. DP).

Figure 5 shows a typical evolution of the optimal value as a function of the computational time (expressed as a percentage of the total time consumed by std. DP). We have chosen three instances from the class 4-piece-b that are the largest in their categories: 5_240, 7_240, and 9_240. PAB is able to provide feasible paths with more than 50% of the optimal cost in less than 5% of the time required by std. DP (to return the optimal solution). Moreover, PAB needs at most 15% of this time to generate feasible paths with a cost between 80% and 95% of the optimal cost. This is observed for all the instances, regardless of the size and the number of pieces of work per duty. Also, PAB has proved its ability to return an optimal solution in a time ranging from 5% to 50% of the std. DP time. Finally, we observe that the descent of the cost is steeper for instances with more d-trips, which means that the algorithm is more efficient when we have to generate longer paths.

These results are especially significant in a CG context, where the ultimate aim is to generate feasible paths with good reduced costs as quickly as possible. Generating paths with more than 80% of the optimal solution in less than 15% of the total time required by std. DP will greatly reduce the CG time.

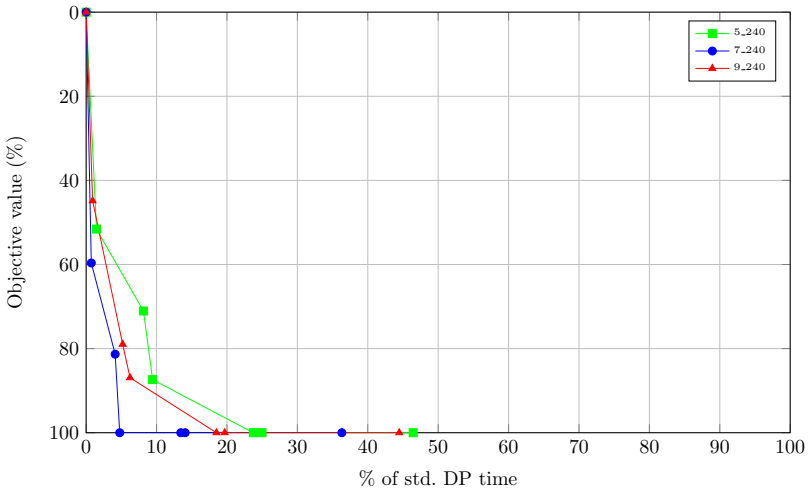

Figure 5: Improvement of objective value as function of % of std. DP time.

5.2.2 Reductions in terms of labels

Tables 5 and 6 give the label results for instances with three or four pieces of work per duty. In these tables, "#Lab." indicates the number of created labels, and "#DCL" indicates the average number of calls to the dominance function per label. This is computed as the ratio of the total number of dominance operations to the number of created labels.

Table 5: Label results for 3-piece-b and 3-piece-e instances.

	b-instances						e-instances					
Inst.	std. DP		MDDPA		PAB		std. DP		MDDPA		PAB	
type	#Lab.	#DCL	#Lab.	#DCL	#Lab.	#DCL	#Lab.	#DCL	#Lab.	#DCL	#Lab.	#DCL
5_120	7.2E+05	18.4	2.5E+05	13.5	3.1E+05	6.7	2.8E+05	5.3	2.4E+03	5.7	2.3E+03	2.6
5_160	1.7E+06	25.0	5.6E+05	19.9	6.0E+05	10.0	7.6E+05	9.7	1.3E+05	13.3	8.0E+04	4.6
5_200	3.3E+06	33.1	1.2E+06	34.4	1.4E+06	16.4	1.5E+06	14.4	4.6E+05	19.9	3.8E+05	9.8
5_240	5.6E+06	33.4	2.5E+06	31.0	2.4E+06	16.1	2.4E+06	16.0	2.8E+05	13.4	2.9E+05	6.4
7_120	1.6E+06	23.3	5.4E+05	15.0	6.5E+05	7.5	5.4E+05	5.0	3.2E+03	4.4	1.6E+03	3.2
7_160	3.6E+06	27.7	1.9E+06	30.0	1.4E+06	12.5	1.8E+06	13.3	2.3E+05	11.0	2.0E+05	5.0
7_200	7.3E+06	36.9	3.3E+06	41.3	3.2E+06	19.8	3.7E+06	17.2	6.1E+05	21.1	4.7E+05	4.9
7_240	1.4E+07	44.4	5.5E+06	44.6	4.3E+06	16.0	5.2E+06	17.5	6.4E+05	14.2	5.9E+05	5.4
9_120	3.0E+06	25.3	1.1E+06	20.2	1.0E+06	8.7	1.3E+06	8.8	1.0E+05	5.3	1.2E+05	3.6
9_160	7.2E+06	32.5	2.5E+06	26.5	1.8E+06	12.2	3.3E+06	13.4	6.6E+05	12.8	6.5E+05	7.3
9_200	1.4E+07	43.7	5.5E+06	36.5	5.0E+06	20.5	7.7E+06	27.8	1.2E+06	12.6	1.4E+06	9.5
9_240	2.4E+07	46.2	1.1E+07	49.7	7.8E+06	19.9	1.1E+07	20.5	1.3E+06	17.7	6.1E+05	4.9

Table 6: Label results for 4-piece-b and 4-piece-e instances.

Inst.	b-instances						e-instances					
	std. DP		MDDPA		PAB		std. DP		MDDPA		PAB	
type	#Lab.	#DCL	#Lab.	#DCL	#Lab.	#DCL	#Lab.	#DCL	#Lab.	#DCL	#Lab.	#DCL
5_120	1.3E+06	30.34	3.6E+05	20.45	8.4E+05	8.91	4.9E+05	8.82	2.4E+03	5.98	2.8E+03	2.69
5_160	3.1E+06	42.05	9.8E+05	29.45	1.7E+06	15.16	1.3E+06	14.44	1.7E+05	14.12	3.3E+05	9.19
5_200	7.4E+06	66.39	2.1E+06	58.00	4.2E+06	25.78	2.6E+06	20.05	7.7E+05	23.36	1.4E+06	17.93
5_240	1.1E+07	62.46	4.4E+06	51.03	6.9E+06	24.40	4.2E+06	25.36	3.8E+05	17.10	8.3E+05	11.40
7_120	2.9E+06	37.11	8.3E+05	22.25	1.8E+06	10.68	9.8E+05	8.65	3.2E+03	4.40	2.5E+03	3.35
7_160	6.4E+06	42.87	2.6E+06	40.69	4.0E+06	19.68	3.1E+06	18.27	3.1E+05	12.24	7.2E+05	9.23
7_200	1.6E+07	71.98	6.1E+06	84.34	1.0E+07	41.60	6.1E+06	23.89	7.3E+05	21.88	1.5E+06	13.46
7_240	2.6E+07	75.23	9.5E+06	71.96	1.4E+07	29.60	8.9E+06	26.86	9.8E+05	20.19	1.7E+06	9.52
9_120	5.6E+06	41.26	1.7E+06	28.78	3.1E+06	13.89	2.4E+06	15.28	1.7E+05	8.70	1.9E+05	3.60
9_160	1.4E+07	55.13	4.5E+06	42.17	6.1E+06	18.93	6.3E+06	21.36	1.3E+06	18.85	2.2E+06	11.72
9_200	3.0E+07	81.24	9.6E+06	80.04	1.5E+07	35.86	1.3E+07	38.38	1.9E+06	18.72	4.4E+06	14.86
9_240	4.0E+07	69.85	1.8E+07	82.72	2.7E+07	39.03	1.8E+07	30.63	2.5E+06	18.76	1.9E+06	8.63

Compared to std. DP, there is a large reduction in the number of labels, which highlights the effectiveness of PAB and explains the reduction in the computational time. The label reduction factor (LRF) is between 1.47 and 2.29 for the 4-piece-b instances, and it is up to 4.02 for the 3-piece-b instances, with an average of 2.78. These ratios are significantly higher for the e-instances, giving an average overall LRF of 48.57.

Figure 6 shows the evolution of the number of labels generated by PAB as a function of time. We have chosen the instances used for Figure 5 (5_240, 7_240, and 9_240). The time axis represents the time consumed by PAB as a percentage of the total std. DP solution time. The isolated marks at 100% indicate the number of labels generated by std. DP, while the linked marks indicate the cumulative number of labels generated during the PAB solution process. The largest number of labels is generated during the Extension step for the final iteration: it represents about 50% of the total number of labels created by PAB. Since the search spaces explored at each PAB iteration are disjoint (see Corollary 3), only the saved labels are kept in memory from one iteration to the next, while the remaining labels are safely deleted. This leads us to conclude that the memory used by PAB in a given iteration is in the worst case equal to half of the total memory required during the solution process. Consequently, we must multiply the LRF by about two to obtain the real memory reduction factor of PAB.

A direct consequence of this is the reduction in the number of calls to the dominance function. Large reduction factors have been observed for all the instances. The dominance reduction factor varies between 2.63 and 6.68 for 4-piece-b instances and between 4.25 and 10.71 for 3-piece-b instances. These ratios are much greater for e-instances, with an average reduction factor of 115.36. This is because the number of calls to the dominance function at a given node is in the worst case a quadratic function of the number of labels at that node.

Tables 5 and 6 highlight the efficiency of PAB compared to std. DP and MDDPA in terms of the average number of dominance operations per label. PAB reduced this rate from 32.5 to 13.9 and from 56.3 to 23.6 on average for 3-piece-b and 4-piece-b instances respectively, and the corresponding values for MDDPA were 30.2 and 51.0. Similar behavior is observed for the e-instances. PAB reduced the value from 14.1 to 5.6 for 3-piece-e instances and from 21.0 to 9.6 for 4-piece-e instances, and the corresponding values for MDDPA were 12.6 and 15.4. We conclude that although MDDPA reduced the number of labels and consequently the number of calls to the dominance function, the average number of dominance calls per label was only slightly affected by this reduction. In contrast, PAB achieved a further reduction ranging between 2 and 3 in the average number of dominance operations per label. This shows that classifying labels by their degree of adjacency and prioritizing dominance between labels with the same degree is better for the solution of SPPRC. These remarkable reductions greatly reduce the computational complexity, which explains the significant time improvement achieved by PAB.

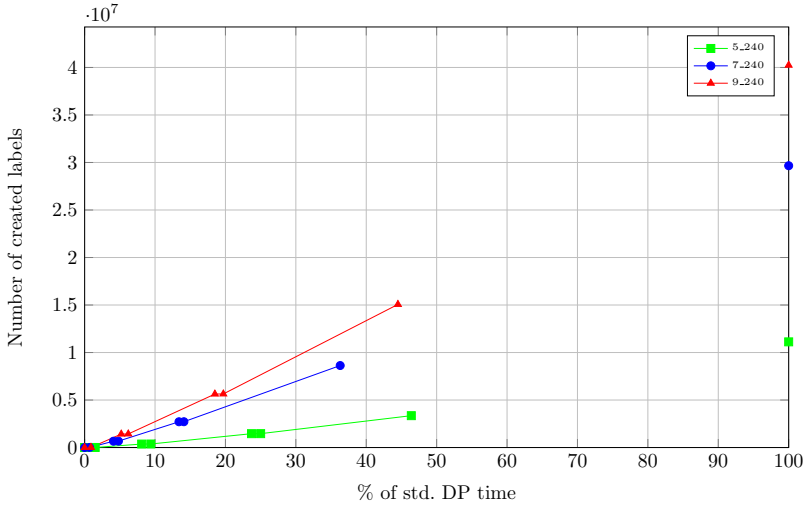

Figure 6: Evolution of the number of labels created by PAB.

5.2.3 Impact of Combination step

Another interesting feature of our approach is the impact of the Combination step. The Combination step quickly improves the objective value of the subproblem between iterations, and it gives the largest part of the cost decrease. We recall that the main role of the Combination step, as mentioned in the last section, is to affinely combine previously generated paths to create new paths with better costs.

Table 7 gives the results for the PAB Combination step for different classes of instances. The columns "% time" give the time for the Combination step as a percentage of the total PAB solution time, and the columns "% gain" indicate the percentage of the optimal value that is due to Combination steps. To compare the performance of the Combination and Extension steps, one can extract the results for the Extension step from Table 7. These values are equal to the complementary part of the percentages for the Combination step, for each attribute.

Table 7: Details of Combination step.

	b-instances				e-instances			
Inst.	3-piece-b		4-piece-b		3-piece-e		4-piece-e	
type	% time	% gain	% time	% gain	% time	% gain	% time	% gain
5_120	21.74	72.69	18.73	69.73	44.41	100.00	45.27	100.00
5_160	20.77	79.47	15.53	79.69	35.42	97.14	32.14	97.14
5_200	16.43	69.31	12.34	67.41	28.67	95.97	23.59	95.97
5_240	14.13	71.01	10.64	67.26	32.05	85.87	29.86	85.87
7_120	20.61	71.75	17.54	69.56	44.58	100.00	49.22	100.00
7_160	16.14	78.32	12.48	83.23	34.81	90.86	33.59	90.86
7_200	13.91	66.63	10.66	69.86	28.52	83.49	22.89	83.49
7_240	13.42	70.36	9.25	69.19	32.31	73.24	29.97	73.24
9_120	19.40	75.49	15.44	75.49	39.45	84.72	42.94	84.72
9_160	17.87	83.03	12.41	83.60	31.21	93.54	29.01	91.24
9_200	12.63	69.59	10.36	69.46	24.83	78.32	22.79	78.32
9_240	11.49	67.91	7.38	65.76	30.27	78.90	26.45	78.90

Table 7 reveals that, for both b- and e-instances with either three or four pieces of work, the Combination step has better performance than the Extension step. For b-instances, the combination time ranges between 7.4% and 21.7% with an average of 14.6% of the total PAB solution time. This percentage decreases slightly with the number of pieces, for all types of instances. In contrast, the extension time often represents more than 78.3% of the PAB time. For e-instances, the total time for the Combination step is about 33.1% on

average for all types of instances. The largest improvement in the objective value (between 65.7% and 83.6% for b-instances and 73.2% and 100% for e-instances) was obtained by the Combination step.

In summary, for b-instances, on average 72% of the optimal value requires only 15% of the time, while 85% of the time is spent finding the remaining 28% of the optimal value. For e-instances, the Combination step requires on average 33% of the PAB time to return 88% of the optimal value, while the remaining 12% is obtained by the Extension step in 67% of the time.

An explanation of these results is that, unlike the Extension step, the Combination step is performed in relatively small subnetworks that are full of primal information. The numbers of labels and dominance operations for the Combination steps are low compared to their values in the Extension steps. Since the main contribution of this step is to affinely combine existing paths to produce new ones (Proposition 7), we conclude that the computational results clearly support our theoretical assertions and justify the use of affine combinations.

Since all the paths generated by the Combination step are affine combinations of previously generated paths, it is clear that no improvement could occur via the Combination technique if there were no good paths previously found by the Extension technique. PAB is consequently an intelligent combination of two techniques (combination and extension) that complement each other, resulting in a primal method that efficiently solves the SPPRC.

6 Conclusion

In this paper, we have considered the SPPRC. The proposed PAB algorithm is a new approach based on an iterative exploration of the search space. Our polyhedral study allowed us to take advantage of some properties of the problem. In particular, we use the notion of adjacency to restrict the search process to a limited space. In addition, using the concept of affine combinations, we have shown that better paths can be easily generated by combining existing paths.

We evaluated our method on VCSPs taken from the literature. A comparison with the standard DP method has indicated the performance of our approach. PAB reduces the solution time for all the test instances, with the reduction factor varying between 2 and 5 on average. As a primal method, PAB converges to optimal solutions faster than MDDPA does, and proves their optimality earlier than MDDPA for b-instances while being competitive for e-instances. Moreover, PAB has shown its ability to generate good paths with more than 50% of the optimal cost in less than 5% of the total time required by the standard DP approach. This result shows that the PAB algorithm is appropriate to CG method since the aim of the CG subproblems is to find good paths as quickly as possible.

This primal paradigm opens up new interesting research tracks for i) efficiently solving more general SPPRCs with nonlinear extension function, covering thus a wider range of CG applications, and ii) reducing "the curse of dimensionality" encountered in general when solving with DP.

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

[4] 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, Montral, Canada.

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

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

[7] Desrosiers J., Pelletier P., Soumis F. Plus court chemin avec contraintes d'horaires. RAIRO Recherche Oprationnelle 17(4), 357–377 (1983).

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

[9] Di Puglia Pugliese L., Guerriero F. A reference point approach for the resource constrained shortest path problems. Transportation Science 47(2), 247–265 (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. Trans. Sci. 35(3), 286–303 (2001).

[13] Handler G.Y., Zang I. A dual algorithm for the constrained shortest path problem. Networks 10(4), 293–309 (1980).

[14] Himmich I., El Hallaoui I., Soumis F. A Multi-Directional Dynamich Programming Algorithm for the Shortest Path Problem with Resource Constraints. Cahier de GERAD, G–2018–05, (submitted to EJOR in February 2018).

[15] Irnich S., Desaulniers G. Shortest path problems with resource constraints. In: Desaulniers G., Desrosiers J., Solomon M.M. (eds), Column Generation, Springer, Boston, MA, 33–65 (2005).

[16] Lozano L., Duque D., L. Medaglia A. An Exact Algorithm for the Elementary Shortest Path Problem with Resource Constraints. Trans. Sci. 50(1), 348–357 (2016).

[17] Muhandiramge R., Boland N. Simultaneous solution of Lagrangean dual problems interleaved with preprocessing for the weight constrained shortest path problem. Networks 53, 358–381 (2009).

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

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

[20] 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).

[21] Zaghrouti, A., El Hallaoui, I., Soumis, F. Integral simplex using decomposition for the set partitioning problem. Operations Research 62(2), 435–449 (2014).