Les Cahiers du GERAD ISSN: 0711–2440

Primal column generation framework for vehicle and crew scheduling problems

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

G–2018–74

October 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

Primal column generation framework for vehicle and crew scheduling problems

Ilyas Himmich
Issmail El Hallaoui
François Soumis

GERAD & Department of Mathematics and Industrial Engineering, Polytechnique Montréal (Québec) Canada, H3C 3A7

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

October 2018
Les Cahiers du GERAD
G–2018–74

Abstract: The primal adjacency-based algorithm and the multi-directional dynamic programming algorithm are two exact methods that have recently been developed to efficiently solve the shortest path problem with resource constraints. These methods are primal in the sense that they are able to produce sequences of feasible solutions using iterative exploration of the search space. Since the shortest path problem with resource constraints often appears as a subproblem in the solution of vehicle and crew scheduling problems using column generation, we propose a new Primal Column Generation framework that embeds these primal methods in a column generation scheme. The Primal Column Generation performs at each iteration a good cost improvement versus time by solving an appropriate restricted subproblem. The use of a primal approach introduces a self-acting ability and a large degree of flexibility. Computational experiments show that the proposed Primal Column Generation is able to find optimal solutions while reducing the time spent solving the subproblems by factors of up to 7 compared to the standard column generation algorithm. This leads to significant improvements in the overall solution times, with an average reduction factor of 3.5.

Keywords: Column generation, subproblems, shortest path problem with resource constraints, dynamic programming, primal paradigm, vehicle and crew scheduling problem

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 these organizations for their support and confidence.

1 Introduction

One of the most important problems arising in mass transit systems is the scheduling of the available resources in order to cover at a minimum cost a set of predetermined services or trips. Indeed, vehicles and crews are the two main resources considered by transport companies; the way in which they are used has a strong impact on the cost and quality of the service. Vehicle scheduling problems, also known as vehicle routing problems, find a least-cost set of vehicle routes that cover all the required trips and obey specific feasibility rules on, e.g., the vehicle capacity, trip ordering and maintenance requirements. Similarly, crew scheduling problems assign crew members to the vehicle itineraries resulting from the vehicle routing, with the goal of minimizing cost while respecting the requirements imposed by the collective agreements and internal safety regulations.

Vehicle and crew scheduling problems (VCSPs) are complex because of their size and the nature of the various feasibility rules. They give rise to very large mixed integer programming problems that are difficult to solve. The most popular method for these problems is column generation (CG) embedded in a branch and bound search tree. CG takes advantage of the structure of the problem and divides it into a coordinator problem, called the master problem (MP), and one or more column generators, called subproblems (SPs). Linking constraints, such as task-covering constraints, are considered in the MP, while the constraints that are separable by vehicle or crew member are treated in the SPs.

The MP usually corresponds to a set covering or set partitioning problem with additional constraints. It is traditionally solved using LP relaxation, and each SP is a shortest path problem with resource constraints (SPPRC). The SPPRCs are usually solved using dynamic programming (DP). Unfortunately, DP becomes time-consuming and inefficient on real-world instances with networks of hundreds of thousands of arcs and dozens of resource constraints.

Several heuristics have been developed to handle this problem of dimensionality. Most of these techniques select subsets of the labels or dominate on a subset of the resources. Since the state removal is not based on mathematical deductions, the solution obtained by CG for the linear relaxation may be at an unknown distance of optimality. Many solvers implement these heuristics in the first iterations of the CG scheme, when it is easy to find columns with negative reduced costs. Since the number of iterations is not known in advance, it is necessary to monitor the solution process when applying these heuristics. Thus, most modelers tune their solvers with several parameters.

Clearly, it suffices for the SPs to feed the MP with columns with sufficiently negative reduced costs. However, we must ensure global optimality at the end of the solution process. Thus, the goal is use approximate methods to reduce the computational time without affecting the quality of the solution.

The efficiency of the CG process depends on the method used to solve the SPs. We therefore propose to incorporate two exact alternatives to classical DP algorithms, namely, the primal adjacency based (PAB) algorithm and the multi-directional DP algorithm (MDDPA). These methods [11, 12] are based on the following idea: first split the state space into disjoint subspaces; then explore the subspaces iteratively in such a way that each iteration learns from the results of the previous ones. We tested these methods on various instances of the SPPRC and obtained promising results.

The two methods are primal in the sense that they are able to return feasible solutions at different stages while solving the SPs for a given CG iteration. Thus, it is possible to terminate the solution process whenever the current columns are judged to be good enough, without proving optimality. We believe that this approach can dramatically reduce the number of generated states, which accelerates the solution process of the SPs. The premature termination may also save considerable SP computational time.

The local efficiency of these primal methods could be altered in a CG context by side effects related to the quality of the dual solution provided by the MP. The goal of this paper is to demonstrate the global efficiency of these methods within CG. The main contributions are as follows:

1. We propose a general framework called primal column generation (PCG) as an alternative to the standard CG framework. PCG does not rely on parameter settings or human intervention. Moreover, it gives the SP solver the ability to determine when there is no need to continue.
2. The PCG framework is the first implementation of our methods within a general CG scheme. We also propose adaptations related to the initial point and the optimality criteria.
3. We perform tests on instances of the VCSP given by [10]. We solve the linear relaxation using our methods within a PCG framework, and we compare the results with those obtained by standard CG using DP to solve the SPs. The results show the effectiveness of PCG: it reduced both the SP time and the overall solution time.
4. We answer two open questions: Is the local efficiency of our methods the same for all CG iterations? How do side effects influence the performance of our methods within a PCG framework?

The paper is organized as follows. Section 2 presents a detailed definition of the VCSP and a literature review. Section 3 describes the classical CG algorithm for this problem. In Section 4, we introduce our PCG framework. Experimental results are given in Section 5, and Section 6 provides concluding remarks.

2 Vehicle and crew scheduling problem

This section presents a definition of the problem, a classification of the different methods used in the literature to solve it, before introducing its mathematical formulation.

2.1 Problem definition

The VCSP finds the minimum-cost set of vehicle routes and crew schedules that covers a predetermined set of services or trips within a fixed planning horizon. We distinguish two problem classes: the single-depot and the multi-depot VCSP. In the multi-depot version, each depot corresponds to a subset of vehicles and crew members. Different transportation companies may have various additional requirements, so it is difficult to provide a general problem definition.

Without loss of generality, we assume that the set of bus lines is predetermined and each line must be served many times over a one-day horizon to meet the passenger demand. Therefore, with each bus line we associate a set of timetabled trips. Each trip must be covered by exactly one vehicle, and a bus can move without passengers from the end of one trip to the beginning of the next. These empty moves are called *deadheads*. Hence, a vehicle schedule or route is composed of a sequence of trips and deadheads, starting and ending at the same depot.

Each trip is divided into *d-trips* or *tasks*, delimited by stops called *relief points*. A relief point is a location where driver changes are permitted. The set of tasks performed by a driver on the same bus is called a *piece of work*. Drivers may take breaks at relief points between two consecutive pieces of work or at the depot. Therefore, the driver's working day, called a *duty*, is a succession of d-trips and breaks.

The set of all driver duties forms the crew schedule. In a feasible crew schedule each d-trip is assigned to a duty, each duty starts and ends at the same depot, and each duty is performed by exactly one crew member. The duty must also satisfy rules concerning the maximum working time, the number of breaks, the minimum break duration, the total work duration, and the number of pieces of work.

The total cost is the sum of the cost of the vehicle routes, the cost of the duties, and various penalties and fixed costs. The cost of the routes includes the cost of fuel, amortization, and bus repairs. The duty cost is composed primarily of driver salaries and overtime payments.

2.2 Literature review

Even the single-depot VCSP problem is NP-hard [5]. It is traditionally solved sequentially; the vehicle routing problem is first solved to generate vehicle routes, and then the crew scheduling problem is solved to assign the

routes to drivers. This sequential procedure was strongly criticized by [1]. They recommended prioritizing the construction of crew scheduling because the problem is more difficult. An integrated approach, in which the vehicles and crews are simultaneously scheduled, was proposed by [6]. Integrated approaches are more efficient than the sequential method, and there is a growing need to synchronize vehicles and crews in practice.

In this paper, we focus on the simultaneous VCSP with a single depot and a homogenous fleet of vehicles. Henceforth, we use the acronym VCSP to refer to this specific version. It is NP-hard and meets the needs of a wide range of transportation companies in small to medium mass-transit systems.

Several VCSP models and solution methods have been proposed in the literature. Most are based on heuristics that can be classified into three categories [7]: 1. Scheduling the vehicles during the crew scheduling process; 2. Taking crew considerations into account during the vehicle scheduling process and subsequently deriving the crew schedules; 3. Completely integrating the vehicle and crew scheduling.

Methods in the first category are the most popular. They are mostly based on the heuristic procedure proposed by [1]. This heuristic decomposes the problem into three steps, each corresponding to a matching problem. The first step generates a set of pieces of work with a duration less than an upper bound T. The second step combines pairs of these pieces into partial duties, and the third step groups the partial duties to generate feasible complete duties. Finally, the vehicle schedules are deduced by omitting the crew-only arcs. A similar three-phase strategy is presented by [13]. In the first phase, a set of partial crew duties covering the timetabled trips is generated by solving a set covering problem. Based on these partial duties, a minimum cost flow problem is solved in the second phase to determine a set of vehicle schedules. Complete duties are generated in the third phase by reconsidering the available partial duties.

The first contribution in the second category was that of [14]. He derived crew schedules from vehicle schedules by making minor changes to the latter based on the crew costs. The results show a slight decrease in the estimated crew costs. For a detailed review of approaches in the first two categories, see [7].

Full integration of vehicle and crew scheduling was introduced by [6]. They proposed a new integer linear formulation with three sets of constraints: set partitioning constraints for the crew schedules, quasi-assignment constraints for the vehicle schedules, and linking constraints to ensure compatibility between the vehicle and crew schedules. They developed two algorithms: the first uses CG to dynamically generate the crew schedules, and the second generates all possible crew schedules at the start of the process. [9] proposed the first exact solution method for the integrated VCSP. This approach is based on a set partitioning formulation with additional linking constraints. Although the authors used a sophisticated methodology, embedding CG, cuts, and clique generators in a branch-and-bound scheme, only small instances (up to 20 trips) were solved to optimality. They subsequently [10] presented a new set partitioning formulation for the crew schedules, incorporating side constraints for the bus schedules. They constructed the crew schedules using CG in a branch-and-bound scheme and then derived the vehicle schedules in polynomial time. Instances with up to 150 trips were solved to optimality using this approach. A detailed review of the formulations and solution methods of the third category is given by [2] and [8].

2.3 Mathematical formulation

The model for an integrated VCSP depends on the requirements of the specific application. In this section, we present the formulation proposed by [10]. This formulation corresponds to a set partitioning problem with additional constraints. It constructs crew schedules while taking into account vehicle considerations, so that the vehicle schedules can subsequently be derived in polynomial time.

Before giving the formulation, we introduce some notation. Let T be the set of trips and D the set of d-trips. Let H be the set of times $h \in H$ at which a bus must leave the depot to travel to the starting location of a trip. We assume that there are several duty types. We denote by Ω^k the set of all feasible paths p representing duties of type k, where K is the set of duty types. Furthermore, we associate with each path p four binary parameters: a_p^d takes the value 1 if path p covers d-trip d and 0 otherwise; s_p^t takes the value 1 if path p contains travel to the start location of trip t and 0 otherwise; e_p^t takes the value one if path p contains travel from the end location of trip t and 0 otherwise; and q_p^h takes the value 1 if path p contains travel starting at or before time $h \in H$ and ending after h and 0 otherwise. Each vehicle has a fixed cost of

use c, and each duty for path p has operating cost c_p. The binary variables θ_p^k indicate whether or not the duty of type k represented by path p is assigned to a driver, and the integer variable N counts the number of buses used to cover the timetabled trips.

The formulation is as follows:

$$\textit{Minimize} \qquad cN + \sum_{k\in K}\sum_{p\in\Omega^k} c_p\theta_p^k \tag{1}$$

s.t.

$$\sum_{k\in K}\sum_{p\in\Omega^k} a_p^d\theta_p^k = 1 \quad \forall d \in D \tag{2}$$

$$\sum_{k\in K}\sum_{p\in\Omega^k} s_p^t\theta_p^k = 1 \quad \forall t \in T \tag{3}$$

$$\sum_{k\in K}\sum_{p\in\Omega^k} e_p^t\theta_p^k = 1 \quad \forall t \in T \tag{4}$$

$$\sum_{k\in K}\sum_{p\in\Omega^k} q_p^h\theta_p^k \le N \quad \forall h \in H \tag{5}$$

$$\theta_p^k \in \{0,1\} \quad \forall p \in \Omega^k, \forall k \in K \tag{6}$$

$$N \in \mathbb{N} \tag{7}$$

The objective function minimizes the total cost, which is the sum of the costs of the vehicles and the wages of the crews.

Constraints (2) ensure that each d-trip is covered exactly once. Constraints (3) and (4) guarantee respectively that exactly one vehicle arrives at the start location and leaves the end location of each trip. Constraints (3) and (4) are flow conservation constraints for the fleet of vehicles, because only d-trip arcs represent vehicle movements between relief nodes. Finally, constraints (5) compute the number of vehicles in use at each departure time. Since the fixed cost of a vehicle is nonnegative and the variables θ_p, $\forall p \in \Omega$, are binaries, the optimal solution gives the number of vehicles N allowing the completion of all the timetabled trips at a minimal cost while satisfying the crew coverage of the d-trips.

3 Standard column generation

The formulation above assumes that all the feasible duties in Ω are known in advance. However, as the number of timetabled trips grows, the size of the corresponding network increases rapidly, which leads to a large number of feasible duties. It quickly becomes intractable to explicitly generate all the duties, and the resulting model would have a huge number of variables and be difficult to solve. The most popular alternative is CG embedded in a branch-and-bound scheme.

3.1 Overview of CG

CG is based on Dantzig–Wolfe decomposition [3]. It splits the corresponding problem into a MP and one or more SPs, one for each duty type. For VCSPs, the MP is simply the linear relaxation of (1)–(7). It considers only the global linking constraints, while the SPs handle the local constraints related to the feasibility of the vehicle and crew schedules.

CG solves at each iteration a reduced version of the MP, called the restricted master problem (RMP), which involves only a small subset of variables $\Omega' \in \Omega$. The RMP is usually solved using linear programming, which provides, at each iteration, a pair of primal and dual solutions. CG uses the dual variable values to update the reduced costs of the arcs in the SP networks. Formally, let $G^k(V^k, A^k)$ be the network corresponding to duty type k. If $\alpha = \{\alpha^d | d \in D\}$, $\beta = \{\beta^t | t \in T\}$, $\gamma = \{\gamma^t | t \in T\}$, and $\delta = \{\delta^h | h \in H\}$ are the vectors of the dual variables associated with the constraints (2)–(5) respectively, the reduced cost of arc $(i,j) \in A^k, k \in K$ is computed as follows:

$$\bar{c_{ij}} = c_{ij} - \sum_{d\in D} a_{ij}^d\alpha^d - \sum_{t\in T} s_{ij}^t\beta^t - \sum_{t\in T} e_{ij}^t\gamma^t - \sum_{h\in H} q_{ij}^h\delta^h.$$

We note that the constants a_{ij}^d, s_{ij}^t, e_{ij}^t, and q_{ij}^h are defined as in Formulation (1)–(7), except that we replace the path index p by the arc index ij.

The reduced cost of a given path is the sum of the reduced costs of the included arcs. The role of the SPs is to generate new feasible paths with negative reduced costs. These paths are added as columns in the RMP, i.e., the subset $\Omega^{'}$ is augmented, and a new iteration is launched. The algorithm stops when no negative-reduced-cost path can be found, which is consistent with the simplex optimality criterion.

3.2 Standard DP algorithm for SPs

For VCSPs, the SPs are usually instances of the SPPRC. Consider a connected acyclic network $G(V, A)$, where V is the set of nodes, including the source node s and the destination node d, and A is the set of arcs. Let R be the set of resource constraints. In addition to the cost c_{ij}, each arc $(i, j) \in A$ has an $|R|$-dimensional resource consumption vector $(r_{ij}^1, r_{ij}^2, ..., r_{ij}^{|R|})$. Similarly, we associate with each node $i \in V$ a resource interval for each resource $t \in R$. The SPPRC finds a least cost path among all the paths from s to d that satisfy the resource constraints induced by R.

The standard approach for the SPPRC is DP. The basic DP algorithm was devised by [4] as an extension of the well-known Bellman–Ford algorithm. It explores the search space by assigning states to each node. A state in node i corresponds to a subpath from the source node to node i. Each state is represented by a multidimensional vector $l = [C_i, R_i^1, R_i^2, ..., R_i^{|R|}]$, called a *label*, where C_i and $R_i^t, t \in R$ are the total cost and the resource consumption of each resource $t \in R$ over all the arcs on the corresponding partial path from s to i.

The set of labels $\{\mathcal{L}_i, i \in V\}$ is initialized with a trivial label $l_1 = [0, 0, ..., 0]$ at the source node, and empty sets at the other nodes. DP explores the state space by calling $Extension(\mathcal{L}_i, j)$ for each node $i \in V$. This function extends the existing labels at node i to its successor nodes $\{j \in V | (i, j) \in A\}$, checks the feasibility of the new labels, and discards infeasible options. A label is feasible if it corresponds to a subpath that respects the resource constraints. In addition to the feasibility restrictions, a state may be fathomed if it cannot lead to an optimal solution. These decisions are made using $Dominance(\mathcal{L}_i)$. Several dominance rules may be considered depending on the requirements of the problem. The most common dominance rule is given in Definition 1.

Definition 1 *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.*

Algorithm 1: Dynamic Programming algorithm $DP(G, \mathcal{L})$

```
for all i ∈ V do
    Dominance(L_i)
    for all j ∈ V do
        T_j ← Extension(L_i, j)
if L_d ≠ ∅ then
    Build Π from L_d
```

Algorithm 1 gives pseudocode for the standard $DP(G, \mathcal{L})$ procedure. DP methods are exact, able to generate feasible paths, and efficient for small and medium instances. However, their performance relies heavily on the effectiveness of the fathoming techniques used to reduce the size of the state space. The number of states increases rapidly with the size of the problem and in the worst case grows exponentially with the number of resources. This gives rise to large spaces, which are computationally expensive to explore. We present below an alternative approach.

4 Primal column generation framework

This section provides the preliminaries needed to explain our PCG framework for the VCSP. We first outline the framework and then introduce the primal methods we implement within PCG, namely the PAB algorithm, MDDPA using a *Nearest First* (NF) strategy and MDDPA using a *Best First* (BF) strategy. For simplification, we assume in this section that there is a single duty type and hence only one SP.

4.1 General overview

The PCG framework has three components: an RMP, an improved decomposable version of the SP, and a control component to manage the dependencies between them. The three-component structure is illustrated in Figure 1.

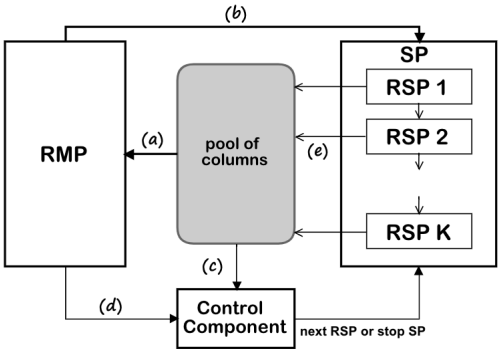

Figure 1: PCG framework

The RMP is similar to that described in the previous section. It is initialized with a small number of columns Ω_0 giving an initial feasible solution. It manages two flows at each iteration: the current dual solution used to update the reduced costs on the arcs in the SP networks (flow(b)), and the set of negative-reduced-cost columns that are sent from the SP to the RMP (flow (a)).

The major enhancements we propose for PCG are related to the solution of the SPs. The framework is based on a decomposition of the SP state space into several disjoint subspaces. Each subspace leads to a restricted subproblem (RSP) that is solved separately. The SP becomes a set of small RSPs, and each subproblem is potentially able to generate feasible paths. These are subsequently transferred to the RMP.

Two decompositions are proposed for the SP state space. The first is based on a measure of distance derived from the *degree of adjacency* of a given path in relation to an initial point, and the second uses the distance of the subpaths from the sink node. These decompositions are explained in Sections 4.2 and 4.3 respectively.

The aim of solving the SP in a CG context is to provide the RMP with sufficiently negative reduced cost feasible paths. The PCG solver does not need to solve all the RSPs in a given CG iteration; it can stop the solution process after each RSP, if a set of good feasible paths has been generated (flow (e)). The quality of the paths is evaluated using the control component of the framework.

The control component determines whether or not the current set of paths should be transferred to the RMP. If the set is not yet adequate, the algorithm continues to the next RSP. To make the decision, the control component uses information about the number and quality of the paths (flow (c)). It also uses information from the RMP (flow (d)), such as the improvement of the objective function between the last two CG iterations and the contribution of the previously added columns to this improvement.

Finally, we emphasize that the PCG does not need any human intervention, moreover, there are fewer parameters to adjust with PCG than with CG. This adjustment is performed at the level of the control component in the beginning of the solution process.

4.2 Multi-directional DP algorithm

We now discuss the state space decomposition and then introduce MPPDA.

4.2.1 MDDPA decomposition

We use the notation of Section 3.2, and we assume in addition that the set of nodes V is topologically ordered, and each node is indexed by its rank in this order. In particular, 1 and $|V|$ denote s and d respectively.

We first build sets of labels $\mathcal{S} = \cup_{i \in V} \mathcal{S}_i$, where each label in $\mathcal{S}_i$ represents a feasible subpath from s to i. These sets are constructed in an initialization step: we first define a restricted subgraph $\bar{G}(\bar{V}, \bar{A})$ of G, where $\bar{V}$ and $\bar{A}$ are the sets of nodes and arcs on the paths for the nondegenerate variables (columns) of the current basic solution of the RMP. Then we call an improved version of DP called the Label Storing Procedure $LSP(\bar{G}, \mathcal{L}, \mathcal{S}, \Pi)$, where $\mathcal{L} = \cup_{i \in V} \mathcal{L}_i$ is the set of active labels initialized with a trivial label ($l_1 = [0, 0, ..., 0]$ at the source node and empty sets at the other nodes) and Π is the set of feasible paths generated by the procedure. We note that $\bar{G}$ may be any connected subgraph of G containing the source and destination nodes. In particular, at the first iteration, $\bar{G}(\bar{V}, \bar{A})$ can be constructed by assigning to each trip a path that starts from the depot, covers the trip, and returns to the depot.

The LSP procedure is performed once at the beginning of the MDDPA. It calls $Dominance(\mathcal{L}_i)$ and $Extension(\mathcal{L}_i, j)$ for each node $i \in \bar{G}$ and fills the set $\mathcal{S} = \cup_{i \in V} \mathcal{S}_i$ with labels that reach nodes not belonging to $\bar{V}$. Formally, let $A^+(\bar{G}) = \{(i, j) \in A \setminus \bar{A}$, such that $i \in \bar{V}$ and $j \notin \bar{V}\}$ be the set of arcs leaving $\bar{G}$. If a label is extended using an arc $(i, j) \in A^+(\bar{G})$, it is stored in $\mathcal{S}_j$. Otherwise, it is added to the set of active labels that may lead to feasible paths in $\bar{G}$. The main steps of LSP are summarized in Algorithm 2.

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

```
S ← ∅
for all i ∈ V do
    Dominance(L_i)
    for all j ∈ V do
        T_j ← Extension(L_i, j)
        if (i, j) ∈ A+(Ḡ) then
            S_j ← S_j ∪ T_j
        else
            L_j ← L_j ∪ T_j
if L_d ≠ ∅ then
    Build Π^k from L_d
    CostBounding(Π, C^best)
Return the set of labels in S_i ∀i ∈ V
```

The second step of the state space decomposition process uses a set of cocycle constraints, defined as follows:

Definition 2 *Consider a directed acyclic network $G(V, A)$. The cocycle constraint of order $k \in \{1, 2, ..., |V| - 1\}$ is the constraint $\sum_{(i,j) \in Co_k} x_{ij} = 1$, where $Co_k = \{(i, j) \in A | i \leq k < j\}$ is the k^{th} cocycle.*

The cocycle constraints for $k \in \{1, 2, ..., |V| - 1\}$ are equivalent to the flow conservation constraints [11]. This means that covering each cocycle exactly once suffices to ensure the connectivity of a given path from s to d.

We denote by i^* the index of the first node whose set of stored labels is nonempty. Formally, $i^* = argmin\{i \in V, \mathcal{S}_i \neq \emptyset\}$. Moreover, each label in $\mathcal{S} = \cup_{i \in V} \mathcal{S}_i$ is denoted by $l = [c_l, r_l^1, r_l^2, ..., r_l^{|R|}]$, where c_l and r_l^t, $t \in R$ are respectively the cost and resource consumptions along the subpath corresponding to l. The resulting SPPRC is as follows:

$$(P_3) \quad Minimize \quad \sum_{(i,j) \in A, i \geq i^*} c_{ij} x_{ij} + \sum_{i \in V, i \geq i^*, l \in \mathcal{S}_i} c_i^l y_i^l \tag{8}$$

s.t.

$$\sum_{i \geq i^*, l \in \mathcal{S}_i} y_i^l = 1 \tag{9}$$

$$\sum_{(i,j) \in Co_k} x_{ij} + \sum_{i > k, l \in \mathcal{S}_i} y_i^l = 1 \qquad \forall k \in \{i^*, i^* + 1, ..., |V| - 1\} \tag{10}$$

$$y_i^l (r_l^t - R_i^t) \leq 0 \qquad \forall i \in V, i \geq i^*, \forall l \in \mathcal{S}_i, \forall t \in R \tag{11}$$

$$x_{ij} (R_i^t + r_{ij}^t - R_j^t) \leq 0 \qquad \forall t \in R, \forall (i, j) \in A, i \geq i^* \tag{12}$$

$$a_i^k \leq R_i^t \leq b_i^t \qquad \forall t \in R, \forall i \in V, i \geq i^* \tag{13}$$
$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A, i \geq i^* \tag{14}$$
$$y_i^l \in \{0,1\} \qquad \forall i \in V, i \geq i^*, \forall l \in \mathcal{S}_i \tag{15}$$

In this model, x_{ij} are the arc flow variables and y_i^l are the label variables that indicate whether or not label $l \in \mathcal{S}_i$ contributes to the construction of an optimal path. R_i^t, $t \in R, i \in V$ are real variables that compute the resources consumed along an optimal subpath from s to i.

Constraints (9) ensure that exactly one label is chosen to construct an optimal path. This may be the trivial label $l_1 = [0, 0, ..., 0] \in \mathcal{S}_1$ normally used by DP algorithms. Constraints (10) cover each cocycle Co_k using either a stored label $l \in \mathcal{S}_i, i \in V, i > k$ or an arc $(i,j) \in Co_k$. Constraints (11) and (12) are resource constraints that update the resource consumptions whenever a new arc or label is selected to be a part of an optimal path. Constraints (13) verify the feasibility of the path in terms of resource constraints. Finally, the binary requirements on variables x_{ij} and y_i^l are expressed by (14) and (15).

This model is a generalization of the classical formulation of the SPPRC that allows the construction of feasible paths using a completion of any label $l \in \cup_{i \in V} \mathcal{S}_i$ and not necessarily the trivial one $l_1 \in \mathcal{S}_1$. Furthermore, every nonempty selection of labels from $\cup_{i \in V} \mathcal{S}_i$ induces a subspace of the entire state space of the SPPRC. If these subspaces are disjoint and complementary, they provide a real decomposition of the whole state space.

MDDPA provides two rules for decomposing the state space. The first classifies stored labels according to their distance from the sink node. The distance of a given label $l \in \mathcal{S}_i, i \in V$ is measured in terms of the number of uncovered cocycles. Labels belonging to the same set of labels therefore have the same distance from the sink node, so the sets of stored labels $\mathcal{S}_i$, $i \in V$ form a decomposition of the state space. However, since there is no guarantee that the extension of these sets of labels will lead to feasible paths and since real-world networks have a huge number of nodes, we construct buckets of labels. Each bucket contains several sets of labels associated with a subset of sequential nodes in the topological order. These buckets are chosen to be disjoint and complementary, so they form a decomposition of the state space based on the distance criterion.

The second rule sorts the stored labels according to their reduced costs. Buckets of labels are constructed in such a way that each contains consecutive labels from a list ordered by reduced cost. Labels in the same bucket are associated with nodes of varying distances from the sink node and labels from the same node may now appear in different buckets. This gives a new decomposition of the space state based on reduced cost.

4.2.2 MDDPA search strategies

For each decomposition, MDDPA provides an appropriate search strategy to explore the induced subspaces. An NF strategy is used for the decomposition based on the distance criterion, and a BF strategy is used for the decomposition based on the reduced cost criterion.

The NF strategy first extends labels that require less computational effort to generate feasible paths, namely those for nodes relatively close to the sink node. Clearly, these labels have fewer uncovered cocycles than those for nodes far from the sink node. Consequently, they need fewer arcs to form complete paths, and hence less computational time. This strategy extends the buckets of labels one at a time in reverse topological order of the nodes. The BF strategy prioritizes the extension of the labels with the most negative reduced costs. This allows the most promising labels to be extended first, thus producing interesting paths as soon as possible. This strategy extends at each iteration the bucket that contains the most important labels in terms of reduced cost.

We extend the labels by calling a DP algorithm in the corresponding restricted subspace. At the end of each iteration, the feasible paths found are considered by the control component. If they are inadequate, a cost bounding is carried out to reduce the size of the subsequent restricted subspaces. The $CostBounding(C^{best})$ procedure uses the best reduced cost C^{best} found in the previous iterations to update the cost upper bounds in the nodes of the network.

Algorithm 3 presents pseudocode for MDDPA embedded in PCG. For illustration purposes and with no loss of generality, we denote by $\mathcal{P}_k, k \in \{1, 2, ..., K\}$ the sets of buckets of labels constructed using either of the two decompositions, where k is the index of these buckets. We denote by Ω^r the pool of negative-reduced-cost columns to send to the RMP in a given CG iteration r. The procedure for the RMP is denoted $RMP(\Omega^r, x^r, \alpha^r)$, where x^r and α^r are respectively the primal and dual solutions returned at iteration r. $DP(G, \mathcal{L})$ is the DP procedure applied to the RSP induced by the bucket of labels $\mathcal{P}_k$. These labels are used to initialize the set of active labels $\mathcal{L}$. Finally, $CCP(\Omega^r)$ is the control component procedure that returns 1 if the existing set of columns is judged to be sufficient, and 0 otherwise. Algorithm 2 provides a unified framework for the application of the NF and BF strategies.

Algorithm 3: Primal Column Generation using MDDPA

```
1: //Initialization //
2: Find an initial solution solution x^0, α^0
3: r ← 1
4: repeat
5:    //Solve the SP//
6:    Ω^r ← ∅ ; C^best ← ∞ ; k ← 1 ; L_1 ← {l_1}; L_i ← ∅ ∀i ∈ V \ {1}
7:    Update arc reduced costs using α^{r−1}
8:    Construct Ḡ using the columns of the basic solution x^{r−1}
9:    Run LSP(Ḡ, L, S, Π^0)
10:   Construct P_k, k ∈ {1, 2, ..., K} from S_i, i ∈ {1, 2, ..., |V|}
11:   repeat
12:      L_i ← S_i , ∀i ∈ V such that S_i ∈ P_k
13:      Run DP(G, L)
14:      if L_d ≠ ∅ then
15:         Build Π^k from L_d
16:         Ω^r ← Ω^r ∪ Π^k
17:         if CCP(Ω^r) = 1 then
18:            break
19:         CostBounding(Π^k, C^best)
20:      k ← k + 1
21:   until k = K
22:   //Solve the RMP//
23:   Ω ← Ω ∪ Ω^r
24:   Run RMP(Ω, x^r, α^r)
25:   r ← r + 1
26: until Ω^r = ∅
27: Return x^r
```

4.3 Primal adjacency-based algorithm

We first define the state space decomposition used by the PAB algorithm and then explain how the algorithm works.

4.3.1 PAB decomposition

The PAB decomposition is based on the notion of *adjacency* between two paths, and between one path and a set of paths in the solution space of the SPPRC. Adjacency is a well-known linear programming notion, defined as follows:

Definition 3 *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 $\boldsymbol{x^2}$.*

For the SPPRC, every feasible path corresponds to an extreme point of the solution space. [12] have provided a new definition of adjacency between two paths in a network using the notion of a *detour*.

Definition 4 *Let π be a path. A set of arcs $\mathcal{D}$ is called a detour if it is compatible with π and minimal, in the sense that none of its strict subsets is compatible.*

A set of arcs $\mathcal{D}$ is said to be *compatible* with a path π if $\mathcal{D}$ is able to replace a subset of the arcs composing π to produce a new complete path π'. The notion of *detour* is illustrated in Figure 2 reproduced from [12].

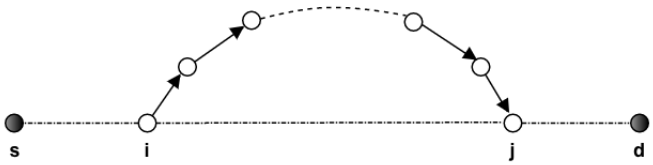

Figure 2: The notion of *detour*

Let π and π' be two paths and A^{π} and $A^{\pi'}$ their respective sets of arcs. The next proposition combines the notion of adjacency and detour.

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

This proposition has been generalized in [12] in two ways. The notion of *k-adjacency* is developed to refer to the degree of adjacency of a given path in relation to either another path or a set of paths. Hence, a path π' is said to be *k-adjacent* to a path π if and only if there are k different detours $\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_k$ allowing the construction of π' from π, i.e., $A^{\pi'} \setminus A^{\pi} = \cup_{i=1}^{k} \mathcal{D}_i$. Similarly, a path π is said to be *k-adjacent* to a set of paths Π if and only if there are k different detours $\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_k$ such that $A^{\Pi} \setminus A^{\pi} = \cup_{i=1}^{k} \mathcal{D}_i$ where A^{Π} is the set of arcs composing all the paths of Π.

[12] proved that every set of paths coincides with a face in the solution space. The dimension of this face depends on the number of affinely independent elements in the set of paths. Consequently, given an initial set of paths Π_0, the degree of adjacency is a useful way to compute the distance of any path π to the face engendered by Π_0. Additionally, this measure provides a new decomposition of the state space of the SPPRC. Each degree of adjacency has a corresponding subspace of the state space, and these subspaces are disjoint. This decomposition is the fundamental pillar of the PAB algorithm.

4.3.2 PAB algorithm

The PAB algorithm is an intelligent combination of DP and the polyhedral properties described above. It solves the SPPRC using iterative calls of DP in restricted state subspaces related to different degrees of adjacency. From a polyhedral point of view, given an initial set of paths Π_0, PAB looks for negative-reduced-cost extreme points in the neighborhood of the face corresponding to Π_0. This neighborhood is sequentially enlarged as the degree of adjacency is increased.

Recall that the degree of adjacency to Π_0 is the number of allowed detours from Π_0. [12] added a new resource called the *adjacency resource*. This resource counts the number of times a subpath leaves Π_0, which fits with the number of detours. In a given iteration k, the upper bound of the adjacency resource is set to k at every node to prevent the extension of subpaths with a degree of adjacency greater than k.

We propose a simple method that avoids the use of the adjacency resource. In contrast to MDDPA, which uses LSP just once at the beginning of the process, PAB manages the flow of created labels using a dynamic call to LSP. The process is as follows: given an adjacency degree k and an initial set of paths Π_0, LSP performs a DP search in the subspace of degree k and returns a set of feasible paths that are k degrees distant from Π_0. Additionally, all labels of degree $k+1$ whose extension was stopped because of the adjacency degree are saved in the sets $\{\mathcal{S}_i, i \in V\}$. These labels are then extended at iteration $k+1$, which ensures a dynamic update of the set of saved labels.

Similarly to MDDPA, the initial point Π_0 is constructed using the paths corresponding to the nondegenerate basic columns extracted from the basic solution of the RMP. These paths have zero reduced costs, so minor changes may be able to produce the desired negative-reduced-cost paths. Such useful changes are exactly what PAB aims to find using detours.

[12] proposed a different initial point for PAB for crew scheduling. They initialize Π_0 with a set of artificial paths that start from the depot, cover a given trip, and return to the depot. This structure allows them to deduce a stopping criterion, namely the maximum degree of adjacency that a feasible path may have in relation to Π_0.

We propose a generalized stopping criterion that does not depend on the structure of the initial point. Our implementation increases the degree of adjacency sequentially and stops whenever the sets of stored labels are empty. The next proposition verifies the accuracy of the algorithm with this stopping criterion.

Proposition 2 *For a PCG iteration, if $\mathcal{S} = \emptyset$, the solution returned by the PAB algorithm is optimal for the SP.*

Proof. If $\mathcal{S} = \emptyset$, this means that there is no label whose extension has been prevented by the PAB algorithm because of violation of the degree of adjacency. Thus, the degree of adjacency is no longer restrictive. Consequently, PAB becomes similar to the DP algorithm, and the solution of the final iteration is optimal. □

Algorithm 4 presents pseudocode for PAB embedded in PCG.

Algorithm 4: Primal Column Generation using PAB

```
//Initialization //
Find an initial solution solution x^0, α^0
r ← 1
repeat
    //Solve the SP//
    Ω^r ← ∅ ; C^best ← ∞ ; k ← 1 ; L_1 ← {l_1}; L_i ← ∅ ∀i ∈ V \ {1}
    Update arc reduced costs using α^{r−1}
    Construct Ḡ using the columns of the basic solution x^{r−1}
    repeat
        Run LSP(Ḡ, L, S, Π^k)
        if Π^k ≠ ∅ then
            Ω^r ← Ω^r ∪ Π^k
            if CCP(Ω^r) = 1 then
                break
        L ← S
        k ← k + 1
    until S = ∅
    //Solve the RMP//
    Ω ← Ω ∪ Ω^r
    Run RMP(Ω, x^r, α^r)
    r ← r + 1
until Ω^r = ∅
Return x^r
```

Remark 1 *No label is generated more than once by the PAB algorithm.*

Remark 1 shows that no redundant work is done by PAB, so no subspace is invoked more than once. Moreover, similarly to MDDPA, the best reduced cost found in a given PAB iteration is used to fathom nonpromising labels to tighten the subsequent subspaces. These two features allow the algorithm to reduce the complexity of DP by reducing the number of created labels.

5 Computational experiments

In this section, we assess the usefulness of our PCG framework, carrying out a computational study where we solve the linear relaxation of VCSP instances. We compare the standard CG algorithm to the new PCG framework using MDDPA with the NF strategy, MDDPA with the BF strategy, and PAB. We begin by describing our test instances.

5.1 Test instances

The VCSP test instances correspond to acyclic networks and were randomly generated using the VCSP generator described in [10]. The complexity of a VCSP depends on the size of the instance, the number of resources, and the width of the resource intervals.

The size of an instance is measured in terms of the number of d-trips to cover, i.e., $tr(rp+1)$, where tr is the number of trips and rp is the number of relief points per trip. We set the number of trips to 120, 160, 200, or 240, and there are 5 or 7 relief points. Each pair (rp, tr) leads to an instance type denoted rp_tr. We generated 5 instances of each type by varying the seed number, for a total of 40 instances. We classify them into 5_rp and 7_rp instances (see Table 1).

Table 1: List of test instances (PCG)

5_rp instances					7_rp instances				
Type	No.	nodes	arcs	d-trips	Type	No.	nodes	arcs	d-trips
5_120	1	50334	78493	720	7_120	1	98395	151488	960
	2	55667	86437	720		2	109052	167422	960
	3	47906	74815	720		3	92990	143343	960
	4	51661	80459	720		4	100960	155311	960
	5	47882	74742	720		5	93059	143398	960
5_160	1	92251	142502	960	7_160	1	180189	275631	1280
	2	97623	150501	960		2	191268	292192	1280
	3	88244	136442	960		3	173005	264803	1280
	4	92235	142451	960		4	180803	276516	1280
	5	86939	134534	960		5	169533	259635	1280
5_200	1	144942	219962	1200	7_200	1	280186	427164	1600
	2	154597	237237	1200		2	304154	463103	1600
	3	136573	210147	1200		3	267271	407720	1600
	4	142169	218646	1200		4	279205	425726	1600
	5	139090	214042	1200		5	272429	415569	1600
5_240	1	211558	323950	1440	7_240	1	414189	629747	1920
	2	218536	334388	1440		2	429705	652996	1920
	3	194907	299016	1440		3	381497	580745	1920
	4	202792	310874	1440		4	398211	605857	1920
	5	199094	305235	1440		5	389186	592206	1920

We consider 7 resource constraints: the minimum and maximum number of pieces of work in the duty, the minimum and maximum length of each piece of work, the length of the duty, the length of breaks, and the total work time in the duty, i.e., the time spent driving or waiting for a bus. We consider duties with up to 2 pieces of work and duties with up to 4 pieces of work. The lower and upper bounds of the resource constraints are given in Table 2, which is reproduced from [10].

Table 2: Work rules for a driver schedule

	Minimum	Maximum
No. of pieces	1	2 or 4
Piece length (min)	15	300
Duty length (min)	45	600
Work time (min)	30	480
Break time (min)	15	90

The experiments were conducted in a computer with an Intel Core i7 3.40 Ghz processor and 16 GB of memory running LINUX. Our PCG framework is implemented in C++, and the LP solver is IBM ILOG CPLEX 12.8.0.0. We use the standard DP algorithm provided by Boost (a well-known C++ library) to solve the SPs.

5.2 Computational results

The computational results are reported in Tables 3 and 4 for instances with 5 and 7 relief points, respectively. For each instance type (column 1), the test instances are numbered from 1 to 5 in column 2. Then, for each algorithm and each instance, we give the number of CG iterations (Itr.), the total time spent solving the SPs (SP time), the total solution time (T time), and finally the total number of columns generated (Col.); all times are in seconds. We refer to the standard CG algorithm as stdCG and to the PCG algorithms as PCG-NF, PCG-BF, and PCG-PAB.

Table 3: Computational times for 5_rp instances

Instance		stdCG				PCG-NF				PCG-BF				PCG-PAB			
Type	No.	Itr.	SP time	T time	Col.	Itr.	SP time	T time	Col.	Itr.	SP time	T time	Col.	Itr.	SP time	T time	Col.
5_120	1	167	1347.0	1498.0	40083	145	419.9	605.2	39848	119	354.1	502.1	26067	490	414.7	628.3	26380
	2	242	835.3	950.9	40445	262	472.4	673.0	44891	298	528.8	692.0	35204	518	567.0	736.3	24988
	3	155	768.2	871.7	37674	121	261.0	395.1	36808	110	236.1	328.6	26069	382	264.7	418.3	21802
	4	187	1260.1	1392.4	40125	195	438.3	651.4	43583	161	429.6	600.9	32863	510	592.2	805.0	25090
	5	286	2008.0	2309.0	50012	201	665.6	944.7	48580	199	829.7	1106.5	42893	665	826.2	1180.3	32050
5_160	1	258	6665.6	7236.1	60388	231	1299.4	2263.7	68240	181	1344.7	1938.5	41665	699	1361.7	2282.1	38932
	2	358	3968.2	4439.8	75705	314	1686.1	2272.0	64844	317	1758.1	2372.7	57118	823	1859.9	2580.8	44887
	3	242	3960.5	4419.4	59324	195	1372.9	2007.0	63333	189	1093.5	1507.5	43029	633	1047.2	1587.8	34594
	4	418	12985.3	13946.9	80177	336	4798.5	5806.4	76893	287	3331.1	4083.6	53774	861	3296.4	4693.9	42517
	5	372	8574.5	9770.6	75757	306	3233.0	4788.7	90342	310	2624.0	3795.9	57786	985	2806.8	4240.9	48157
5_200	1	299	17187.1	18966.2	90377	221	3203.8	5359.9	83400	194	3794.4	4931.8	55310	968	3019.6	5899.4	58303
	2	450	20502.6	22213.6	97862	424	7304.4	9930.0	107409	393	7532.0	9309.8	73956	1221	6643.8	9228.7	63897
	3	298	11227.5	12271.6	84270	211	2237.5	3737.0	82224	208	2751.3	3811.0	60437	817	2088.9	3769.5	49298
	4	322	14972.1	16633.3	94451	250	2744.6	5543.3	87144	232	2937.8	4585.9	61106	857	2449.2	4603.2	50269
	5	420	28301.2	31154.3	99950	465	16217.1	19353.2	107408	408	12765.5	15407.7	78996	1280	9457.1	13558.5	61841
5_240	1	479	91559.3	96529.7	126701	393	17073.1	21755.9	116714	324	22385.9	25891.8	89556	1368	10891.3	18514.5	76347
	2	637	46902.7	50990.1	129190	423	9121.8	13629.6	114619	426	11140.7	14589.4	86229	1358	8783.9	14284.1	75236
	3	396	34909.9	39033.4	123348	299	6777.5	11753.0	119501	272	6681.5	10467.8	80513	1278	6327.2	12210.0	71790
	4	289	13134.3	15728.4	96234	240	3842.8	7236.0	99949	211	3913.0	5864.1	66070	896	3283.8	6747.1	61921
	5	282	37346.1	38623.8	132222	268	23206.3	24787.0	137447	223	15842.0	17162.3	104168	973	6838.1	8516.5	88295

For the 40 instances considered, the PCG framework was faster than stdCG. The primal algorithms greatly reduced both the SP time and the total time. To show this, we compute the total time reduction factors as a ratio of the total stdCG time to the total PCG time using each of the three primal algorithms. For the 5_rp instances, PCG-NF reduced the total time by a factor of between 1.41 to 4.44, with an average ratio of 2.57. For PCG-BF, these ratios range between 1.37 and 3.85 with an average of 2.85. Similarly, PCG-PAB reduces the total time by a factor of between 1.29 and 5.21, with an average of 2.80. For the 7_rp instances, PCG-NF reduced the total time by a factor of up to 6.18, with an average of 3.07. The ratio is about 3.29 on average for PCG-BF, and PCG-PAB was slightly more efficient with an average ratio of 3.40.

These improvements are mainly due to the huge reductions in the SP times. To clarify this, we define the SP time reduction factor for each primal algorithm as the ratio of the SP time for this algorithm to the SP time for stdCG. For the 5_rp instances, PCG-NF and PCG-BF give an overall average SP time reduction factor of 3.53 and about 3.49 respectively. For PCG-PAB, these factors are between 1.47 and 8.41, with an overall average of 4.10. The values grow significantly with problem size. For the 7_rp instances, PCG-NF reduces the SP time by a factor of up to 7.13, with an average of 3.82, while PCG-BF has an average reduction factor of 3.85. For PCG-PAB, the factor reaches 9.69, with an overall average of 5.46. Figure 3 gives, for each instance type, the SP time reduction factors realized by our three primal algorithms in a PCG framework. Each factor is computed as the average of the SP times for the five instances of this type.

Table 4: Computational times for 7_rp instances

Instance		stdCG			PCG-NF				PCG-BF				PCG-PAB			
Type	No.	Itr.	SP time	T time	Col.	Itr.	SP time	T time	Col.	Itr.	SP time	T time	Col.	Itr.	SP time	T time	Col.
7_120	1	256	7676.4	8463.4	69686	186	1661.1	2512.3	67002	138	1306.8	1745.3	39141	693	1540.4	2408.9	40495
	2	336	4022.1	4421.1	63965	326	1771.7	2564.4	70352	316	1838.4	2423.2	54368	751	2190.4	2780.8	41247
	3	219	2950.5	3410.4	61852	124	779.7	1139.0	55375	132	626.4	877.7	34747	481	697.8	1191.2	30402
	4	269	6409.9	6887.4	67347	195	1369.5	2012.3	60706	187	1826.5	2333.8	46434	685	1551.8	2105.3	39050
	5	314	6966.7	8083.0	77798	249	2305.9	3085.9	68698	216	2172.6	2848.1	51268	892	2991.7	4478.9	49951
7_160	1	329	29108.3	31622.3	99634	252	5320.1	8728.6	103850	201	4918.3	6277.1	57058	924	3605.3	6553.3	55805
	2	544	26181.3	28299.4	126027	324	7863.0	9804.4	97071	336	5619.4	7710.6	78268	1074	5419.9	8243.0	67497
	3	287	14921.7	16588.1	92484	225	4848.1	6847.4	87344	196	3410.2	4677.6	55074	743	2631.3	4122.7	49566
	4	524	52612.6	57084.2	126860	351	12714.2	16824.3	111255	310	13136.0	16961.4	81371	1234	10535.8	15994.9	66880
	5	401	28761.5	33176.3	115099	342	12077.2	16622.7	125054	324	10348.7	13572.6	86999	1234	7625.5	13445.6	68221
7_200	1	317	66294.5	72890.8	130037	281	13888.2	19435.8	127378	209	12794.2	15669.1	76588	1166	8024.4	18021.4	84573
	2	710	134298.0	145517.0	174340	442	30803.6	39477.8	136850	446	34301.5	41871.0	112559	1574	24229.1	36536.6	94121
	3	335	35169.1	39508.8	126936	233	9807.4	13356.2	122581	238	10266.1	12645.4	79869	1080	6930.0	14619.0	76004
	4	334	42611.6	48253.5	135293	294	10647.4	15793.1	120517	242	8282.0	11534.0	81069	1194	7675.2	18422.0	81578
	5	427	102086.0	113160.0	146153	451	54087.4	64479.1	133504	441	61017.5	69857.6	119576	1473	17825.1	34309.4	90534
7_240	1	482	499539.0	518601.0	182595	401	70083.4	83920.9	165695	350	131481.0	142005.0	122630	1618	76686.7	106170.0	107763
	2	710	279066.0	300450.0	199696	478	71804.8	84481.9	158733	522	77824.7	90910.6	131615	1667	28793.6	49037.2	111352
	3	399	105834.0	124081.0	177159	338	32000.0	46618.1	178402	359	49288.8	60045.4	122245	1542	15362.4	37954.8	102502
	4	296	42070.1	48890.2	136887	262	18002.7	25230.4	136910	247	18711.5	23253.2	89473	1165	10167.6	20651.9	89363
	5	323	82134.1	96630.1	125825	256	18753.9	28972.3	113443	226	17809.6	29384.7	80095	1703	11713.4	29507.9	82245

Another important feature of the PCG framework is the substantial reduction in the SP time as a percentage of the total time. Tables 3 and 4 show that the SP times often represent more than 85% of the total time (the average is about 90%) when stdCG is used. The average value is about 70% for PCG-NF, is about 76% for PCG-BF, and does not exceed 65% for PCG-PAB. This is partially explained by increased MP times, but the increases are insignificant compared to the huge reductions in the SP times.

PCG also reduces the number of generated columns. This reduction was not significant for PCG-NF, but it appears clearly with the other two algorithms. PCG-BF reduces the number of columns by 27% on average for 5_rp instances and 34% on average for the 7_rp instances. PCG-PAB generates on average at most 60% of the number of columns generated by stdCG.

Furthermore, Tables 3 and 4 show that MDDPA reduces the number of CG iterations for almost all the instances. For the 5_rp instances, PCG-NF reduces the number of CG iterations by an average of 15% and PCG-BF by 22%. For the 7_rp instances these values are 23% and 28%. However, PAB increases the number of CG iterations by, on average, a factor of 3.5.

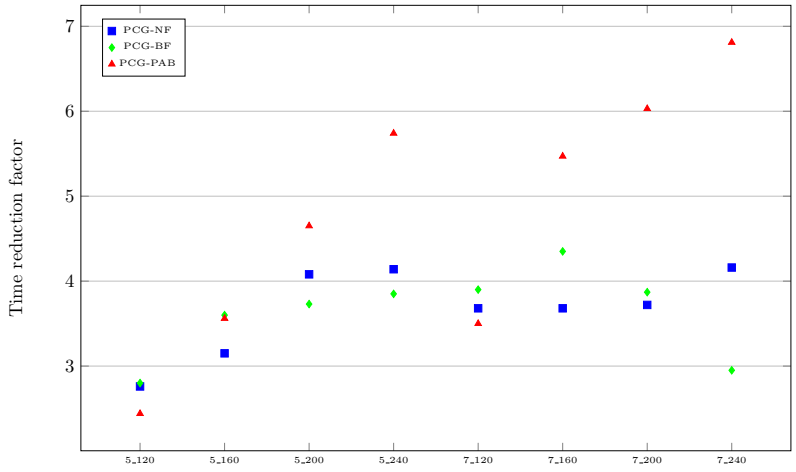

Figure 3: SP time reduction factors

Table 5 gives the average MP time and SP time per iteration for each instance type. This table reveals a remarkable characteristic of PCG-PAB: its ability to perform small improvements in less time than stdCG, PCG-NF and PCG-BF. For the 5_*rp* instances, the average SP time per iteration for PCG-PAB is about 13 times less than that for stdCG and 5 times less than that for both PCG-NF and PCG-BF. These reduction factors are much greater for the 7_*rp* instances: the SP time per iteration is reduced by average factors of 19, 6, and 7 compared to stdCG, PCG-NF, and PCG-BF respectively. Moreover, PCG-PAB reduced the MP time per iteration of stdCG by a factor of 2 for the 5_*rp* instances and about 3 for the 7_*rp* instances.

Table 5: Average SP and MP times per iteration (in seconds)

Instance	MP time per iteration				SP time per iteration			
type	stdCG	NF	BF	PAB	stdCG	NF	BF	PAB
5_120	0.7	1.1	1.0	0.4	6.0	2.5	2.7	1.0
5_160	2.1	3.4	2.7	1.2	21.5	8.6	7.8	2.5
5_200	5.0	8.1	5.7	2.5	50.9	17.6	19.2	4.3
5_240	9.4	11.7	9.5	3.9	106.2	38.1	41.9	6.0
Average	4.3	6.1	4.7	2.0	46.2	16.7	17.9	3.5
7_120	2.3	3.2	2.5	1.1	20.3	7.4	8.0	2.4
7_160	7.3	10.6	8.2	3.3	72.1	27.7	26.6	5.4
7_200	18.3	18.9	14.7	8.6	174.0	63.5	70.8	9.4
7_240	46.8	34.0	30.7	12.7	418.2	112.3	163.3	18.0
Average	18.7	16.7	14.0	6.4	171.2	52.7	67.2	8.8

PCG-PAB focuses its search on small neighborhoods of the current basic solution. This allows it to quickly find good negative-reduced-cost columns without investing a large computational effort in the SPs. Consequently, it sends only a few columns to the MP. The dual values are then rapidly updated, which ensures fast convergence of the overall PCG algorithm in a larger number of iterations. The PCG-NF and PCG-BF algorithms are also primal, but they require higher computational times to return good columns because they explores larger subspaces. In summary, PCG-PAB is faster but with small improvements, while PCG-NF and PCG-BF are slower but with larger improvements. This can be clearly seen in Figure 4.

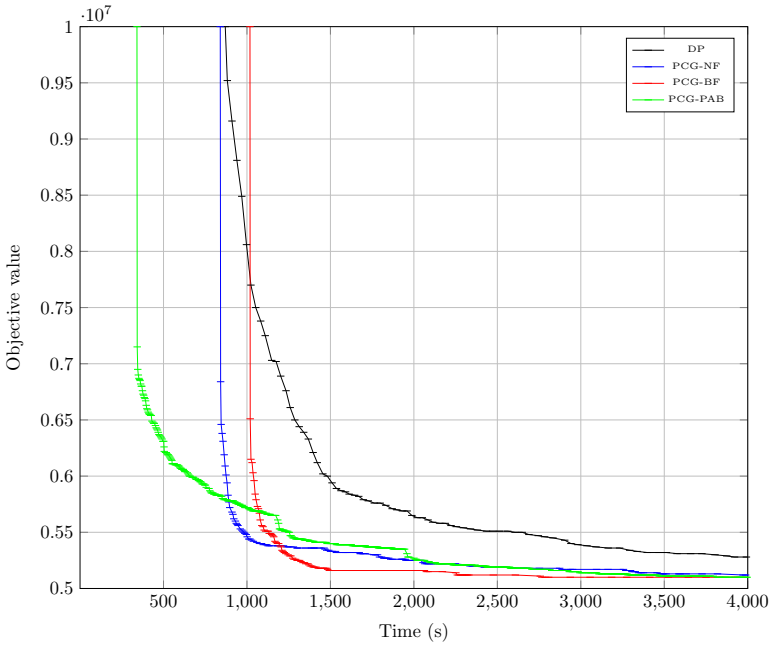

Figure 4: Improvement of the objective value over time

Figure 4 is based on data from one 7_*rp* instance; it tracks the evolution of the objective value (cost) over time for all the algorithms. For clarity, it focuses on the region with the most important differences between the algorithms. For the selected instance, the total time reduction factor was 4.04 for PCG-PAB, 3.75 for PCG-NF, and 4.65 for PCG-BF.

PCG outperforms stdCG in terms of the speed of convergence, regardless of the algorithm used to solve the SPs. In particular, until 1500 s, the cost achieved by stdCG is at least three times greater than the objective value reached by any of our algorithms. This ratio then decreases until it becomes approximately 2. Moreover, the graph shows that PCG-NF and PCG-BF decrease the objective value in a similar way. PCG-NF is initially faster, and then PCG-BF becomes more efficient. PCG-PAB gives a slow and continuous decrease of the objective value, and the decrease starts earlier than that for PCG-NF and PCG-BF.

PCG-PAB makes small cost decreases early, providing the best solution quality at this stage. It is then overtaken by PCG-NF and, later, PCG-BF. PCG-PAB is able to generate good columns at the beginning of the CG because it performs an adjacency-based search in the neighborhood of the current basic solution. PCG-NF and PCG-BF have better behavior once the dual values start to stabilize. For all the algorithms, the cost decrease is slow toward the end of the process.

To conclude this section, we observe that the PCG framework is remarkably efficient compared to the classical CG framework. The time reduction factors are large, and the reductions in the number of columns are also considerable, especially for PCG-PAB. The PAB algorithm is initially the best algorithm, and later the NF and BF strategies may be alternated within MDDPA.

6 Conclusion

In this paper, we have provided a new PCG framework for the VCSPs. This work is the first attempt to embed into a CG scheme recently developed primal algorithms for the SPs, namely PCG-PAB, PCG-NF, and PCG-BF. We focused on the skeleton of the PCG framework, the underlying primal paradigm, and the state space decompositions used. Furthermore, we adapted the new primal methods to the CG context by introducing heuristic and optimal stopping criteria that allow the CG to quickly and intelligently find negative-reduced-cost columns. The PCG is an innovative framework for CG algorithms, it gives the SP solver a large degree of flexibility and autonomy, which improves the overall performance. Our experiments show that the PCG framework outperforms standard CG, producing optimal solutions for all the instances in shorter computational times. In particular, the SP times were reduced by factors of up to 7.

We plan to apply this work to airline scheduling problems, where the SPPRC arises as a pricing SP inside CG. We also plan to investigate the strengths of the three primal methods in order to find a way to combine them in a CG process. Finally, our long-term goal is to develop efficient new learning techniques that could benefit from the flexibility of our PCG framework and help the solver to focus on the regions in the SP state spaces with the most potential.

References

[1] Ball, M., Bodin, L., Dial, R. A matching based heuristic for scheduling mass transit crews and vehicles. Transportation Science 17, 4–31 (1983).

[2] Borndörfer, R., Löbel, A., Weider, S. Integrierte Umlauf und Dienstplanung im Nahverkehr. ZIB Report 02–10, Konrad-Zuse Zentrum, Berlin (2002).

[3] Dantzig, G.B., Wolfe P. Decomposition principle for linear programs. Operations Research 8(1), 1–157,(1960).

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

[5] Fischetti, M., Martello, S., Toth, P. The fixed job scheduling problem with working-time constraints. Operations Research 37, 395–403 (1989).

[6] Freling, R., Boender, G., Paixao, J.M.P An integrated approach to vehicle and crew scheduling. Technical Report 9503-A, Econometric Institute, Erasmus University Rotterdam (1995).

[7] Freling, R., Huisman,D., Wagelmans, A.P.M. Models and algorithms for integration of vehicle and crew scheduling. Journal of Scheduling 6, 63–85 (2003).

[8] Grötschel, M., Borndörfer, R., Löbel, A. Duty scheduling in public transit. Jäger, W. (Ed.), Mathematics – Key Technology for the Future, Springer, Berlin, 653–674 (2003).

[9] Haase, K., Friberg, C. An exact branch and cut algorithm for the vehicle and crew scheduling problem. N. H. M. Wilson (Ed.), Computer-Aided Transit Scheduling, Springer Verlag, Berlin, Germany, 63–80 (1999).

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

[11] Himmich, I., El Hallaoui, I., Soumis, F. A multidirectional dynamic programming algorithm for the shortest path problem with resource constraints. Cahier de GERAD, G-2018-05, submitted to EJOR in February 2018.

[12] Himmich, I., Ben Amor, H., El Hallaoui, I., Soumis, F. A primal adjacency-based algorithm for the shortest path problem with resource constraints. Cahier de GERAD, G–2018–09, submitted to Trans. Sci. in Feb. 2018.

[13] Patrikalakis, I., Xerocostas, D. A new decomposition scheme of the urban public transport scheduling problem. In M. Desrochers and J. M. Rousseau (Eds.), Computer-Aided Transit Scheduling: Proceedings of the Fifth International Workshop, Springer Verlag, Berlin, 407–425 (1992).

[14] Scott, D. A large linear programming approach to the public transport scheduling and cost model. In J. M. Rousseau (Ed.), Computer Scheduling of Public Transport 2, North Holland, Amsterdam, 473–491 (1985).