Computers & Operations Research 156 (2023) 106248

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor

A branch-and-bound algorithm for the Precedence-Constrained Minimum-Cost Arborescence problem

Mauro Dell'Amico, Jafar Jamal, Roberto Montemanni *

University of Modena and Reggio Emilia, Reggio Emilia, Italy

ARTICLE INFO

Keywords:
Precedence constrained arborescences
Mixed integer linear programming
Branch and bound
Lagrangian relaxation
Network optimization

ABSTRACT

The Precedence-Constrained Minimum-Cost Arborescence problem, in which precedence constraints are enforced on pairs of vertices, has been recently proposed. The purpose of the constraints is to prevent the formation of directed paths along the tree that violate a precedence relationship. The problem has been shown to be NP-hard, and formulations for the problem have been proposed in the literature.

This work introduces a branch-and-bound algorithm based on a Lagrangian relaxation for solving the problem. The results show that the newly proposed method is 74.6% faster, on average, compared to the state-of-the-art methods recently available in the literature.

1. Introduction

The *Minimum-Cost Arborescence problem* (MCA) is a well-studied problem in the area of graph theory. The objective of the problem is to find a directed minimum-cost spanning tree in a directed graph rooted at some vertex r called the root. Chu and Liu (1965), and Edmonds (1967), independently proposed the first polynomial time algorithm for solving the problem. The most efficient implementation of the algorithm was later on proposed by Gabow et al. (1986). The Minimum-Cost Arborescence problem can be formally described as follows. A directed graph $G = (V, A)$ is given where $V = \{1, \ldots, n\}$ is the set of vertices, $r \in V$ is the root of the arborescence, and $A \subseteq V \times V$ is the set of arcs with a cost c_{ij} associated with every arc $(i, j) \in A$. The objective of the problem is to find a minimum-cost directed spanning tree in G rooted at r, i.e. a set $T \subseteq A$ of $n - 1$ arcs, such that there is a unique directed path from r to every other vertex $j \in V \backslash \{r\}$ in the subgraph induced by T.

Variations of the MCA with different objective function and/or constraints have appeared in the literature since the MCA was first proposed. The *Constrained Arborescence Augmentation problem* (Li et al., 2017) can be described as follows. Given a weighted directed graph $G = (V, A)$ and an arborescence $T = (V, A_r)$ rooted at $r \in V$, the objective of the problem is to find a subset of arcs A' from $A - A_r$ with the sum of the weights of the arcs in A' is minimized, and there still exists an arborescence in the graph $G' = (V, A_r \cup A' - a)$ for each arc $a \in A_r$. The *Maximum Colorful Arborescence problem* (Fertin et al., 2017) is an NP-hard problem (Fertin et al., 2017) that is described as follows. Given a weighted directed acyclic graph, a set of colors C, with each vertex in the graph having a specified color from C, the objective of the problem is to find a maximum-weight arborescence in which no color appears more than once. The *Resource-Constrained Minimum-Weight Arborescence problem* (Fischetti and Vigo, 1997) is another NP-hard problem (Fischetti and Vigo, 1997) which can be described as follows. Given a weighted directed graph with each vertex associated with a finite resource, the objective of the problem is to find a minimum-cost arborescence such that the sum of the outgoing arcs from each vertex in the arborescence is at most equal to the resource associated with that vertex. The *Capacitated Minimum Spanning Tree problem* (Gouveia and Lopes, 2005) is another NP-hard problem (Gouveia and Lopes, 2005) that can be described as follows. Given a root vertex $r \in V$ and an integer Q, with each vertex $j \in V \backslash \{r\}$ is associated with a non-negative integer demand q_j, the objective of the problem is to find a minimum spanning tree rooted at r, such that any subtree off the root r, the sum of the weights of the vertices of that subtree is at most Q. Other interesting variations of the MCA problem can be found in Cai et al. (2004), Frieze and Tkocz (2021) and Morais et al. (2019).

A variation of the MCA, named the *Precedence-Constrainted Minimum-Cost Arborescence problem* (PCMCA), where a set of precedence constraints is included between pairs of vertices, was proposed by Dell'Amico et al. (2021). More formally, given a set R of ordered pairs of vertices, for each precedence $(s, t) \in R$ any path in the arborescence that includes both s and t must visit s before visiting t. The objective of the problem is to find an arborescence of minimum total cost that satisfies the precedence constraints. A mathematical programming model was also discussed in the paper. Other Mixed

* Corresponding author.
E-mail address: roberto.montemanni@unimore.it (R. Montemanni).

https://doi.org/10.1016/j.cor.2023.106248
Received 1 September 2022; Received in revised form 15 December 2022; Accepted 11 April 2023
Available online 14 April 2023

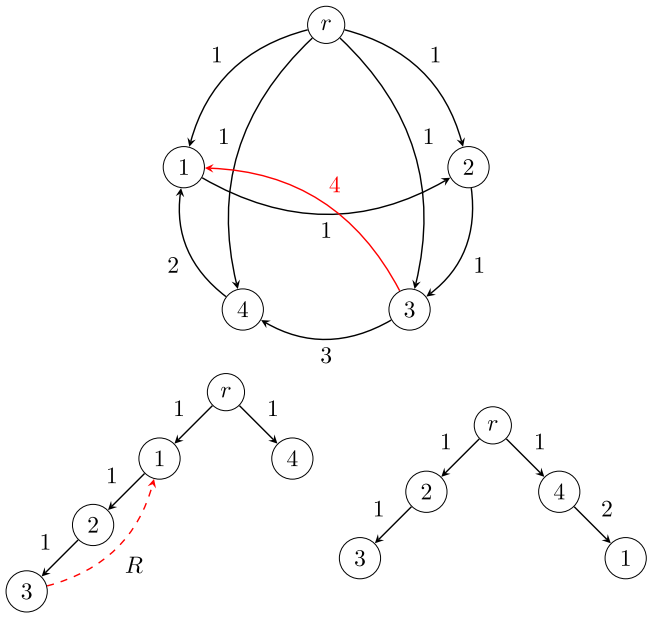

Fig. 1. Comparing a MCA and a PCMCA solution. The graph on top is the instance graph with its respective arc costs, with the precedence relationship $(3,1) \in R$ highlighted in red. The graph on the bottom left shows the optimal MCA, whereas the graph on the bottom right shows the optimal PCMCA. The MCA solution is not a feasible PCMCA solution since vertex 1 precedes vertex 3 on the same directed path and $(3,1) \in R$.

Integer Linear Programming models for the PCMCA were proposed in Chou et al. (2023) and Dell'Amico et al. (2023).

The solving approach we propose is inspired by that discussed in Lucena (1992) and Escudero et al. (1994). In particular, in Escudero et al. (1994) a Lagrangian relax-and-cut approach for the *Sequential Ordering Problem* was introduced. The algorithm proposed works by relaxing some of the constraints in a Lagrangian fashion. This reduces the overall problem to a MCA problem. Later, Toth and Vigo (1995) proposed a Lagrangian relax-and-cut algorithm for the *Capacitated Shortest Spanning Arborescence*, where a Lagrangian relaxation reduced the main problem again to a MCA problem.

Fig. 1 presents an example that shows the difference between the classic MCA and the PCMCA. The example instance graph with its respective arc costs is shown in the figure on the top, with the precedence relationship $(3,1)$ highlighted in red. The figure on the bottom left shows a feasible MCA solution with a cost of 4. The MCA solution is infeasible for the PCMCA since $(3,1) \in R$, and vertex 1 belongs to the directed path connecting r to vertex 3. To make the solution feasible for the PCMCA, vertex 1 must succeed vertex 3 on the same directed path, or the two vertices must reside on two disjoint paths. A PCMCA feasible solution with a cost of 5 is shown on the bottom right of the figure.

The present work proposes a branch-and-bound (B&B) algorithm based on a Lagrangian relaxation for solving the PCMCA based on a Mixed Integer Linear Programming (MILP) model recently proposed in Chou et al. (2023). The rest of the paper is organized as follows. Section 2 discusses a MILP model for the PCMCA. Section 3 introduces a Lagrangian relaxation of the model previously introduced. Section 4 describes the B&B algorithm we propose to solve the PCMCA. Computational results are presented in Section 5, while conclusions are summarized in Section 6.

2. An integer linear programming model

A MILP formulation for the PCMCA was previously proposed in Chou et al. (2023), and is at the basis of the present study. The formulation extends the basic connectivity constraint for the MCA (Houndji et al., 2017), such that it respects the precedence constraint between vertex pairs. Considering a set $S \subseteq V \backslash \{r\}$, a constraint is added for each $k \in S$, which forces that at least one active arc must enter S originating from the set of vertices that are allowed to precede k on the path connecting r to k.

Let x_{ij} be a variable associated with every arc $(i,j) \in A$, such that $x_{ij} = 1$ if arc $(i,j) \in T$ and 0 otherwise, where T is the resulting optimal arborescence. Let $V_k = \{i \in V : (k,i) \notin R\}$ be the set of vertices that can precede k on the same path connecting r to k without introducing a precedence violation. Note that according to the definition $k \in V_k$, since $(k,k) \notin R$. The PCMCA can be formulated as follows.

$$\min \sum_{(i,j)\in A} c_{ij} x_{ij} \tag{1}$$

$$\text{s.t.} \sum_{(i,j)\in A} x_{ij} = 1 \quad \forall j \in V \backslash \{r\} \tag{2}$$

$$\sum_{\substack{(i,j)\in A: \\ i \in V_k \backslash S, j \in S}} x_{ij} \geq 1 \quad \forall k \in V \backslash \{r\}, \forall S \subseteq V_k \backslash \{r\} : k \in S \tag{3}$$

$$x_{ij} \in \{0,1\} \quad \forall (i,j) \in A \tag{4}$$

Constraints (2) enforce that every vertex $v \in V \backslash \{r\}$ must have a single parent. Constraints (3) model the connectivity constraints, that is every vertex $k \in V \backslash \{r\}$ must be reachable from the root. The set of constraints (3) is the core of the model. The inequalities ensure the connectivity of the arborescence as they reduce to the classical connectivity constraints for the MCA which are $\sum_{(i,j)\in A: i \notin S, j \in S} x_{ij} \geq 1 \, \forall S \subseteq V \backslash \{r\}$ when the set R of precedence relationships is empty. This is true because $V_k = V$ for all $k \in V \backslash \{r\}$ when R is an empty set. Constraints (3) also impose the precedence relationships, which imply that every vertex $k \in V \backslash \{r\}$ must not be reachable from the root r through vertex t if $(k,t) \in R$. Finally, Constraints (4) define the domain of the variables.

Going back to constraints (3), they impose the precedence relationships as follows. A *violating path* is defined as a path which contains both s and t, but visits t before visiting s for some $(s,t) \in R$. If a candidate solution contains a violating path, then by removing t and its incident arcs from the solution, then s is no longer reachable from the root. Inequalities (3) will enforce s to be reachable from r through the vertices that are allowed to precede s (i.e. V_s). Fig. 2 shows an example of how a violating path is resolved. The details can be found in the caption of the figure.

The precedence-enforcing set of constraints (3) is what makes the problem NP-hard compared to the MCA, which is solvable in polynomial time (Edmonds, 1967). Therefore, reducing the set of constraints (3) to the classical connectivity constraints only, as explained earlier, would reduce the problem to a MCA and make the problem easier to solve. Furthermore, an optimal solution of the relaxed model provides a lower bound on the optimal solution of the PCMCA instance.

3. A Lagrangian relaxation

In this section we present a relaxation of the PCMCA obtained by relaxing the precedence-enforcing constraints (3), and leaving only the classical connectivity constraints of the MCA, and thus making the dual problem solvable in polynomial time.

Formally, as explained in Section 3, the set of constraints (3) can be split into two sets. The first set enforces that every vertex is reachable from the root (connectivity constraints). Conversely, for any vertex s that is part of a precedence relationship (i.e. $\exists (i,s) \in R$), the second set of constraints enforces that vertex s is reachable from the root through a path not containing vertex t if $(s,t) \in R$ (precedence-enforcing constraints). Splitting constraints (3) into the two sets of

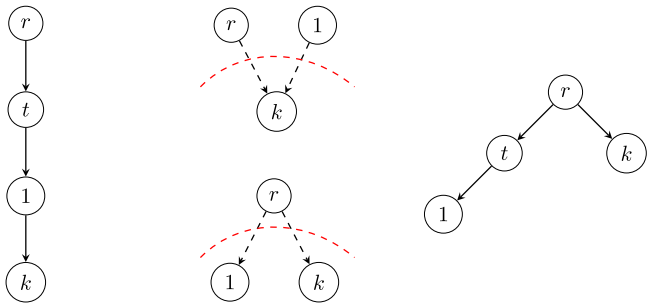

Fig. 2. Example on how constraints (3) resolve violating paths. The figure on the left is a candidate solution that contains a violating path from t to k. The two figures in the middle show the two possible inequalities (3) for vertex k, where the vertices under the red dashed line is the set of vertices S, and the vertices above the dashed red line is the set of vertices $V_k \backslash S$. The dashed black arrows in the figure are the set of arcs that could be part of a feasible solution. The figure on the right shows one possible feasible solution after enforcing constraints (3).

constraints results in the following model for the PCMCA.

$$\min \sum_{(i,j)\in A} c_{ij} x_{ij} \tag{5}$$

$$\text{s.t.} \sum_{(i,j)\in A} x_{ij} = 1 \quad \forall j \in V\backslash\{r\} \tag{6}$$

$$\sum_{\substack{(i,j)\in A:\\ i\notin S, j\in S}} x_{ij} \geq 1 \quad \forall S \subseteq V\backslash\{r\} \tag{7}$$

$$\sum_{\substack{(i,j)\in A:\\ i\in V_k\backslash S, j\in S}} x_{ij} \geq 1 \quad \forall k \in V\backslash\{r\} : \exists (s,k) \in R, \forall S \subseteq V_k\backslash\{r\} : k \in S \tag{8}$$

$$x_{ij} \in \{0,1\} \quad \forall (i,j) \in A \tag{9}$$

Constraints (7) are now the connectivity constraints that enforce every vertex to be reachable from the root. Constraints (8) are the precedence-enforcing constraints which enforce every vertex k to be reachable from the root through a path not containing t if $(k,t) \in R$.

An immediate relaxation of the PCMCA can be obtained by removing the precedence-enforcing constraints (8), thus reducing the problem to a MCA. The MCA can be solved in polynomial time using Edmond's algorithm (Edmonds, 1967) that has a computational complexity of $O(EV)$, or a faster implementation proposed by Gabow et al. (1986) that has a computational complexity of $O(E \log V)$ for sparse graphs, and $O(V^2)$ for dense graphs.

A stronger lower bound can however be obtained by adding constraints (8) to the objective function (5) in a Lagrangian relaxation fashion (see Fisher (1981)).

3.1. The relaxed model

Let $\lambda_S^k \geq 0$ be a Lagrangian multiplier for each $k \in V\backslash\{r\} : \exists (s,k) \in R$ and $S \subseteq V_k\backslash\{r\} : k \in S$. A Lagrangian relaxation of PCMCA can be obtained by introducing the precedence-enforcing constraints (8) in the objective function, with their corresponding Lagrangian multipliers λ.

The following Lagrangian relaxation of the PCMCA can be obtained.

$$\min \sum_{(i,j)\in A} c_{ij} x_{ij} + \sum_{\substack{k\in V\backslash\{r\}:\\ \exists (s,k)\in R}} \sum_{\substack{S\subseteq V_k\backslash\{r\}:\\ k\in S}} \lambda_S^k \left(\sum_{\substack{(i,j)\in A:\\ i\in V_k\backslash S, j\in S}} 1 - x_{ij} \right) \tag{10}$$

$$\text{s.t.} \sum_{(i,j)\in A} x_{ij} = 1 \quad \forall j \in V\backslash\{r\} \tag{11}$$

$$\sum_{\substack{(i,j)\in A:\\ i\in V\backslash S, j\in S}} x_{ij} \geq 1 \quad \forall S \subseteq V\backslash\{r\} \tag{12}$$

$$x_{ij} \in \{0,1\} \quad \forall (i,j) \in A \tag{13}$$

Using a modified cost matrix c' defined as:

$$c'_{ij} = c_{ij} - \sum_{\substack{k\in V\backslash\{r\}:\\ i\in V_k}} \sum_{\substack{S\subseteq V_k\backslash\{r\}:\\ k,j\in S, i\notin S}} \lambda_S^k \tag{14}$$

the objective function (10) can be rewritten as the following.

$$\min \sum_{(i,j)\in A} c'_{ij} x_{ij} + \sum_{k\in V\backslash\{r\}} \sum_{\substack{S\subseteq V_k\backslash\{r\}:\\ k\in S}} \lambda_S^k \tag{15}$$

For any given set of nonnegative multipliers λ, the optimal solution of the Lagrangian relaxation is a valid lower bound for the PCMCA Fisher (1981). Moreover, since the Lagrangian relaxation calls for finding a MCA of the modified cost matrix c', it can be optimally solved in polynomial time using one of the algorithms mentioned earlier.

3.2. Solving the Lagrangian relaxation

In this section we describe a subgradient optimization procedure for the set of Lagrangian multipliers λ.

In our implementation of the subgradient optimization procedure, the set of Lagrangian multipliers λ is passed as a parameter, as it is possible to solve the relaxation starting from an empty or a precomputed set of multipliers, possibly with a non-zero value. Using a set of precomputed Lagrangian multipliers is useful when the procedure is being used inside a B&B algorithm (see Section 4), as the Lagrangian multipliers constructed for a parent node in the search-tree can be passed down to that node's children which in turn could speed up the convergence of the procedure.

Algorithm 1 describes the optimization procedure used for computing values for the set of Lagrangian multipliers λ, and therefore calculating a lower bound for the original PCMCA instance. Let M be a user-defined parameter which represents the maximum number of iterations, and m is the iterations counter, starting from 0. Let x^λ be the optimal solution of the MCA problem based on the modified cost matrix c' relative to λ. Let $\bar{P}$ be the set of all violating paths in x^λ, such that each $p \in \bar{P}$ is a sequence of vertices belonging to a violating path that starts with t and ends with s such that $(s,t) \in R$. Let $\alpha \in \mathbb{R}^+$ be the step size, which is a step taken in the direction of a positive subgradient. Different procedures for computing the step size α are possible and a few of them will be described in Section 5. Let $L(c,\lambda)$ be a function which returns the optimal solution of the Lagrangian relaxation based on cost matrix c and the current set of Lagrangian multipliers λ. Let $Separate(x^\lambda)$ be a function which returns a set of violating paths in the solution x^λ. Finally, let g be the value of a subgradient. The function $Separate(x^\lambda)$ has a computational complexity of $O(|V|^2)$, and can be briefly summarized as follows. Given an arborescence T, for each vertex $s \in V$ we traverse the arborescence T in a backward direction starting from s. If vertex $t \in V$ is reached during the traversal and $(s,t) \in R$, then the path from t to s is added to the set of violating paths returned by the function.

Step 5 finds an optimal solution of the MCA instance based on the objective function (15). Step 6 updates the value of the lower bound as the maximum between the current LB value, and the value of the objective function (15). Step 7 finds the set of all violating paths in the solution x^λ. A path is considered violating if it starts with t and ends with s such that $(s,t) \in R$. Step 9 finds the set of vertices belonging to the set S, which are all the vertices in p without t. Step 10 adds the corresponding Lagrangian multiplier λ_S^s to the set of multipliers λ, with a user-defined non-negative initial value. Steps 13 and 14 update the value of the Lagrangian multiplier λ_S^k. The value of step size $\alpha \in \mathbb{R}^+$ is a user-defined value.

A common practice is to remove the subset of Lagrangian multipliers in λ that had a value of 0 during the last k iterations. This process can be added to Algorithm 1 before optimizing the Lagrangian multipliers at step 12. However, adding this step might not be beneficial when the number of Lagrangian multipliers that get a 0 value is very small

Algorithm 1 Subgradient Method for Solving the Lagrangian Relaxation

procedure ComputeLB(λ, c, M)
  $m = 0$
  $LB = 0$
  do
    $x^\lambda = L(c, \lambda)$
    $LB = \max(LB, c'x^\lambda + \lambda)$
    $\bar{P} = \text{Separate}(x^\lambda)$
    for all $p = \{t, 1, 2, \dots, s\} \in \bar{P}$ **do**
      $S = p \backslash \{t\}$
      $\lambda = \lambda \cup \lambda_S^s$
    end for
    for all $\lambda_S^k \in \lambda$ **do**
      $g = 1 - \sum_{i \in V_k \backslash S, j \in S : (i,j) \in A} x_{ij}^\lambda$
      $\lambda_S^k = \max(0, \lambda_S^k + \alpha g)$
    end for
    $m = m + 1$
  while $\bar{P} \neq \phi$ and $m < M$
end procedure

compared to the total number of multipliers. Some preliminary experiments suggested that this is the case for our problem. We observed only around 10%–15% of the multipliers were going to 0, so the overhead required to periodically check them was not justified. At the same time, the total memory footprint was in fact not substantially reduced. Therefore, we decided to remove multipliers with value 0 only during branching (see Section 4.3).

4. A branch-and-bound algorithm

In this section we describe a branch-and-bound (B&B) algorithm for the exact solution of the PCMCA.

4.1. Data structures

The proposed B&B algorithm contains two main data structures, the *search-tree*, and the *search-tree node*. The search-tree represents the set of all possible MCA solutions of the original instance, but only a subset of those solutions are feasible PCMCA solutions. Note that a feasible PCMCA solution is not necessarily a leaf node in the search-tree, as solving the Lagrangian relaxation might result in a feasible, but suboptimal, PCMCA solution. The search-tree node denoted as v represents a feasible MCA solution or a feasible PCMCA solution and contains information about that solution. The following sections describe the type of information that is stored inside the search-tree and the search-tree node.

4.1.1. Search-tree

The following information is stored inside the search-tree data structure:

- LB: is the current lower bound value for the optimal solution of the PCMCA instance.
- UB: is the current upper bound value for the PCMCA instance, which is the best known solution value, according to objective function (1).
- *SearchTreeNodes*: is a minimum priority queue that contains the set of all search-tree nodes that are currently unexplored in the search-tree.

4.1.2. Search-tree node

The following information is stored inside the search-tree node data structure:

- λ: is the set of Lagrangian multipliers related to this search-tree node.
- c': is the modified cost matrix of the graph G relative to the set of Lagrangian multipliers λ.
- *Pred*: is a vector of size $|V|$, where $V[j] = i$ if i is the parent of j in the optimal MCA solution associated with the search-tree node. To indicate the root of the arborescence we set $V[r] = -1$.
- $\overline{LB}$: is a lower bound value for the optimal PCMCA solution under the search-tree node, that is the value of objective function (15) associated with the solution represented by *Pred*.
- $\bar{c}$: is the reduced cost matrix of G relative to the MCA solution of the search-tree node. The MCA is solvable as a linear program, and thus the reduced costs can be computed.

4.2. Lower bound and upper bound computation

A lower bound on the optimal solution of node v in the search-tree, named $\overline{LB}(v)$, is computed using Algorithm 1, based on the set of Lagrangian multipliers λ and cost matrix c' that are stored in node v. The lower bound on the optimal solution of the PCMCA instance (global lower bound) named LB is equal to $\overline{LB}(v)$, where v is the search-tree node on the top of the priority queue *SearchTreeNodes*.

An upper bound on the optimal solution of the PCMCA instance is found once a node v in the search-tree contains a feasible PCMCA solution (an MCA solution that does not contain a violating path). The UB value is updated if the cost of the solution value is less than the current value of UB.

4.3. Branching scheme

In the proposed B&B algorithm the tree is traversed according to a *best-first search* strategy, that is the node with the lowest lower bound value ($\overline{LB}$) is expanded first. The search-tree node with the lowest lower bound value can be found in constant time as it resides on the top of the priority queue *SearchTreeNodes*, that is stored inside the *Search-tree* data structure. Note that a *depth-first search* strategy, which explored the search tree as far as possible along each branch of the search-tree before backtracking, was considered. However, the computational results obtained while using such a strategy produced an overall increase in the solution time. This was due on the one hand to the slow convergence of the lower bounds, and on the other hand to the worse incumbent solution encountered in the beginning of the search, that led to a less effective pruning and consequently to the exploration of a larger portion of the solution space.

At each node v of the search-tree, let $P' \subset A$ be a violating path that belongs to the solution at node v. Recall that a path is violating if it starts with vertex t and ends with vertex s such that $(s, t) \in R$. For each $(i, j) \in P'$, we create a new search-tree node which forbids arc (i, j), and imposes all the arcs that precede arc (i, j) in the path P'. In our implementation, an arc is forbidden to appear in the solution by setting $c_{ij} = \infty$, and an arc (i, j) is imposed to appear in the solution by setting $c_{kj} = \infty$ for each $k \neq i$. The Lagrangian multipliers related to node v that have a non-zero value, are copied to the newly created nodes from v. Note that this strategy removes, as a side effect, multipliers with value 0. Finally, the search-tree nodes that are created from node v are added to the priority queue after computing their $\overline{LB}$ value, and the value of LB is eventually updated accordingly. Once a search-tree node is expanded, it is removed from the priority queue *SearchTreeNodes*.

Fig. 3 shows an example of how a search-tree node is expanded. Each search-tree node is represented by a rectangle, red arcs indicate forbidden arcs, and blue arcs indicate imposed arcs. In this example, we have the violating path $\{(t, 1), (1, 2), (2, 3), (3, s)\}$. For each arc in the

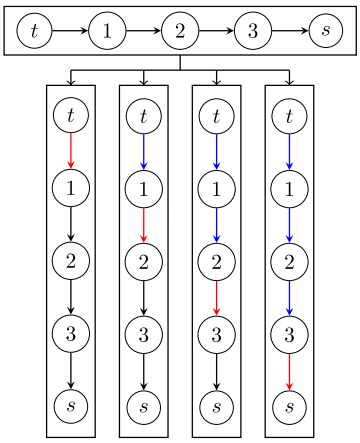

Fig. 3. An example which shows how a search-tree node containing a violating path is expanded into four separate search-tree nodes. Each rectangle represents a search-tree node. Red arrows indicate forbidden arcs, while blue arrows indicate imposed arcs in the solution.

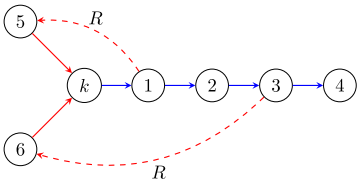

Fig. 4. An example which shows an additional set of arcs that are forbidden to appear in the solution when a certain path rooted at k is imposed in the solution. In this example, dashed red arrows indicate a precedence relationship, blue arrows indicate imposed arcs, and all red arrows indicate forbidden arcs. Arc $(5,k)$ is forbidden to appear in the solution as $(1,5) \in R$ and arc $(k,1)$ is imposed in the solution. Arc $(6,k)$ is forbidden to appear in the solution as $(3,6) \in R$, and a path rooted at k is imposed in the solution that includes vertex 3.

violating path, we forbid that arc to appear in the solution, and all the arcs preceding that arc on the path are imposed to be part of the solution. By doing so, the search-tree node containing a violating path is expanded into $|A'|$ search-tree nodes, where $|A'|$ is the number of arcs in the violating path selected.

4.4. Instance reduction, pruning and bypass rules

Once a new search-tree node is created, it is possible to forbid another set of arcs (other than the ones removed during branching) that creates a violating path as follows. If a directed path $p \subset A$ rooted at $k \in V$ is imposed in the solution, then $c_{ik} = \infty$ for all $(i,k) \in A$ such that $(j,i) \in R$ and j appears in the path p. An example of this is illustrated in Fig. 4. An arc $(i,j) \in A$ is forbidden if $\lceil \overline{LB}(v) + \bar{c}_{ij} \rceil \geq UB$, where $\bar{c}$ is the reduced cost matrix at node v, and UB is the upper bound or the best-known solution. Finally, a search-tree node v can be pruned from the search-tree if $\overline{LB}(v) \geq UB$, or if the node does not contain feasible MCA solution rooted at r.

In the B&B algorithm, a bypass rule is also enforced as follows. If the number of nodes generated in the search tree reaches 2000 nodes, the MILP model introduced in Section 2 is solved for the original problem instance. The bypass rule is enforced in order to avoid the size of the search-tree from growing too large and going out of count, since once the search-tree grows too large, the lower bound on the optimal solution improves very slowly, and feasible solutions are found less and less frequently.

5. Experimental results

The computational experiments for evaluating the proposed B&B algorithm are based on the benchmark instances of TSPLIB (Reinelt, 1991), SOPLIB (Montemanni et al., 2008) and COMPILERS (Shobaki and Jamal, 2015) originally proposed for the Sequential Ordering Problem (SOP) (Escudero, 1988). The benchmark instances are the same instances previously adopted in Dell'Amico et al. (2021), Chou et al. (2023) and Dell'Amico et al. (2023) for PCMCA.

All the experiments are performed on an Intel i7 processor running at 1.8 GHz and equipped with 8 GB of RAM. The B&B algorithm and the MILP model are implemented in C++ 11, and are complied with Microsoft C/C++ Optimizing Compiler v19.

5.1. Lagrangian relaxation

5.1.1. Step size

In the subgradient method there are many different types of step size rules, and the selection of which rule to use can substantially affect the performance of the algorithm (Bazaraa and Sherali, 1981). In this section we compare the convergence of the method using three step size rules from the literature: *constant step size*, *diminishing step size*, and *p-diminishing step size*.

The constant step size is a positive real number that does not change with each iteration.

The diminishing step size is a positive real value which decreases with each iteration. The formula for calculating a diminishing step size is $\alpha_k = \frac{a}{k}$, where α_k is the step size at iteration k, and a is a real positive value.

The p-diminishing step size is similar to the diminishing step size rule, however it does not decrease at each iteration. The step size value decreases every time the value of the objective function decreases compared to the previous iteration. The formula for calculating a p-diminishing step size is $\alpha_k = \frac{a}{p}$, where α_k is the step size at iteration k, a is a real positive value, and p is an integer positive value. The value of p starts at 1, and every time the value of the objective function decreases compared to the previous iteration the value of p is incremented by 1, and the new step size value is calculated using the beforementioned formula.

Fig. 5 compares the value of objective function (15) at the root node for a subset of the representative instances using a constant step size $\alpha_k = 0.1$, compared to using a diminishing step size as explained earlier with $a = 1$, and a p-diminishing step size as explained earlier also with $a = 1$. Each case is run for a total of 100 iterations. The results suggests that using a constant step size, the value of the objective function typically tends to oscillate, which sometimes gives the advantage of escaping a local optima, and the value of the objective function is increased at a faster rate in the first few iterations. On the other hand, using a diminishing step size, convergence is sometimes faster, and feasible solutions are found much quicker compared to using a fixed step size. This in turn helps in pruning the search-tree and reducing the size of the explored solution space. This is the case since the value of the objective function oscillates less frequently compared to a fixed step size, and thus feasible solutions are retrieved with a relatively small optimality gap. The p-diminishing step size rule has a very similar behavior to the diminishing step size rule in most cases, but convergence happens at a higher rate, with feasible solutions found more frequently.

In general, the p-diminishing step size rule is expected to perform better when compared to the other two step size rules, as it behaves similar to a constant step size as long as the value of the objective function is increasing at each iteration, and once the value of the objective function decreases compared to the previous iteration, the step size is decreased which helps to prevent the value of the objective function from oscillating over time.

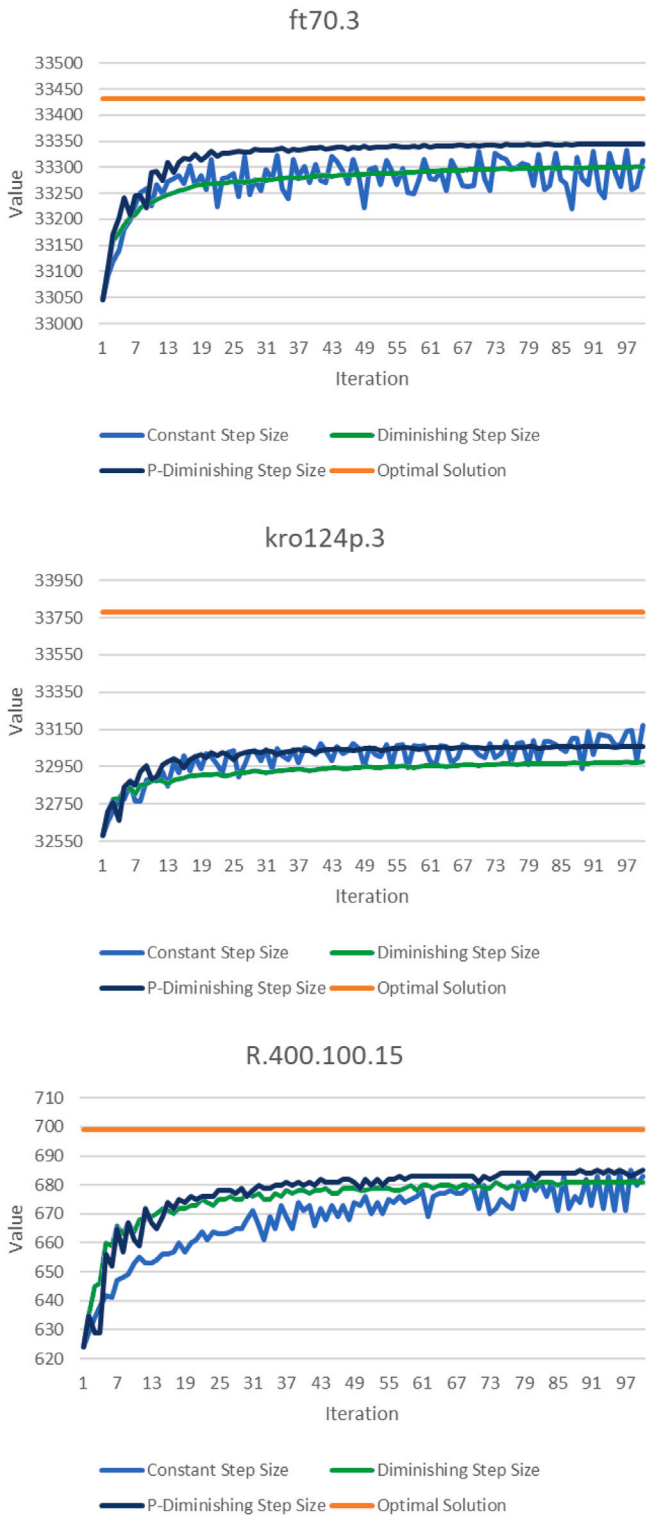

Fig. 5. Comparing the value of the objective function at each iteration using a fixed step size $\alpha_k = 0.1$, diminishing step size $\alpha_k = \frac{1}{k}$, and p-diminishing step size $\alpha_k = \frac{1}{p}$.

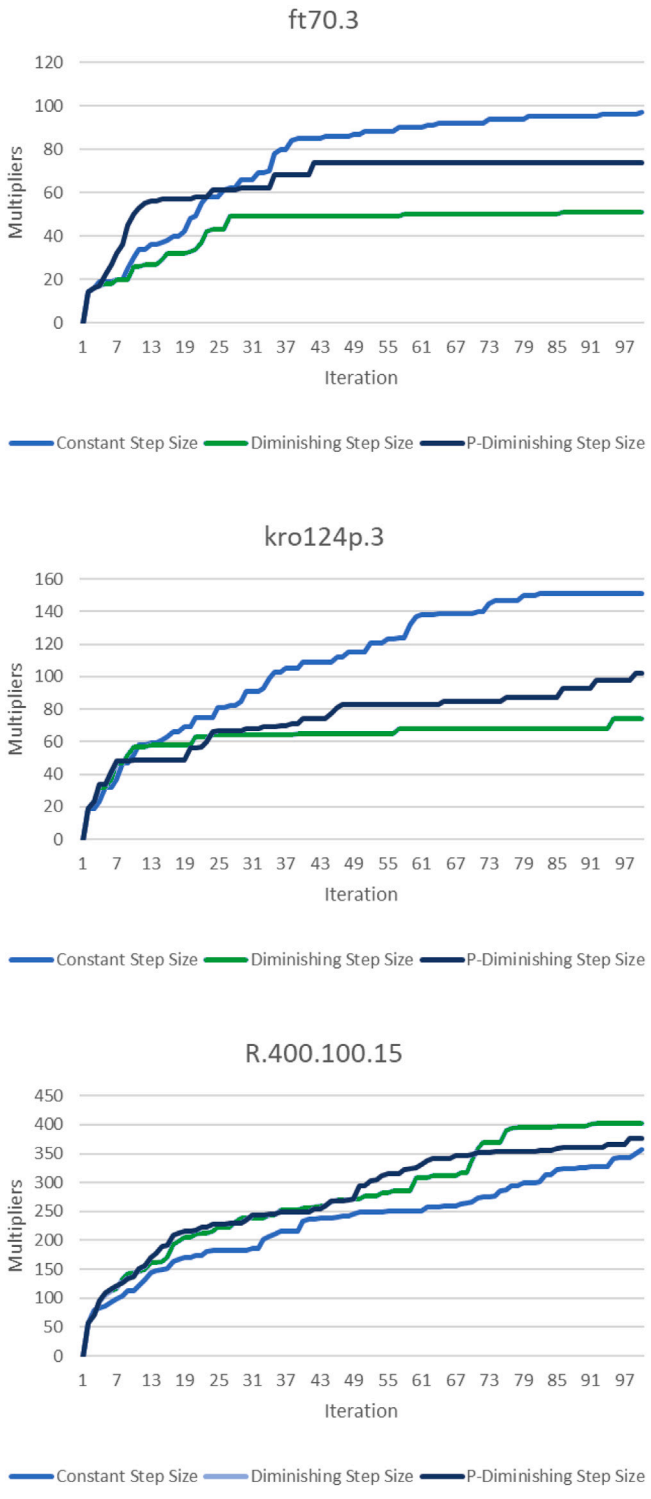

Fig. 6. Comparing the increase in the number Lagrangian multipliers for the root node at each iteration using a fixed step size $\alpha_k = 0.1$, diminishing step size $\alpha_k = \frac{1}{k}$, and p-diminishing step size $\alpha_k = \frac{1}{p}$.

5.1.2. Number of iterations

Increasing the number of iterations M in the subgradient method could potentially tighten the lower bound value computed for a search-tree node. However, that would likely lead to increasing the number of multipliers in the objective function (15), in addition to the MCA being computed at each iteration, and thus increasing the computation time per node. Fig. 5 shows also the value of the lower bound at the rood node of the B&B tree at each iteration for three step size rules of the subgradient method. In the three subfigures, the curve starts to plateau around iteration 20. The value of $M = 10$ is chosen as it balances well

Table 1
Overall computational results for comparing the MILP solver and the B&B algorithm for TSPLIB instances.

Instance					MILP (Chou et al., 2023)		B&B \w $\alpha_k = 0.1$		B&B \w $\alpha_k = \frac{1}{k}$		B&B \w $\alpha_k = \frac{1}{p}$	
Name	$\|V\|$	$\|A\|$	$\|R\|$	z^*	Nodes	Time [s]	Nodes	Time [s]	Nodes	Time [s]	Nodes	Time [s]
br17.10	18	258	48	25	0	0.015	1	0.004	0	0.001	0	0.004
br17.12	18	251	55	25	0	0.016	1	0.004	0	0.006	0	0.006
ESC07	9	50	22	1531	0	0.031	0	0.000	0	0.001	0	0.000
ESC11	13	128	28	1752	0	0.031	0	0.000	0	0.001	0	0.000
ESC12	14	146	36	1138	0	0.016	0	0.001	0	0.001	0	0.001
ESC25	27	640	62	1041	0	0.063	0	0.001	0	0.002	0	0.001
ESC47	49	2225	127	703	0	0.547	1	0.013	1	0.009	0	0.018
ESC63	65	3800	360	56	0	0.218	4	0.095	3	0.036	1	0.052
ESC78	80	5880	440	502	0	0.047	0	0.019	0	0.008	0	0.012
ft53.1	54	2745	117	3917	5	0.375	6	0.051	3	0.028	1	0.021
ft53.2	54	2727	135	3978	55	0.547	19	0.158	11	0.128	8	0.107
ft53.3	54	2540	322	4242	0	0.453	7	0.116	39	0.384	11	0.102
ft53.4	54	1998	864	4882	0	0.047	9	0.064	7	0.096	6	0.067
ft70.1	71	4814	156	32846	0	2.750	2	0.043	2	0.043	1	0.039
ft70.2	71	4783	187	32930	0	2.719	4	0.143	4	0.099	5	0.122
ft70.3	71	4616	354	33431	145	38.250	170	6.622	75	2.607	92	2.789
ft70.4	71	3506	1464	35179	369	6.281	45	0.662	136	2.371	14	0.395
rbg048a	50	1906	544	204	0	0.031	0	0.010	1	0.007	0	0.005
rbg050c	52	2043	609	191	0	0.047	3	0.034	3	0.031	1	0.011
rbg109	111	6662	5548	256	0	0.094	0	0.025	0	0.019	0	0.017
rbg150a	152	12317	10635	373	1	0.219	0	0.041	0	0.028	0	0.031
rbg174a	176	16496	14304	365	1	0.313	1	0.083	0	0.044	0	0.046
rbg253a	255	34082	30688	375	0	1.125	0	0.162	0	0.162	11	0.636
rbg323a	325	56451	48849	754	0	1.047	0	0.265	0	0.204	0	0.235
rbg341a	343	62320	54986	610	0	3.031	13	1.706	7	0.634	5	0.579
rbg358a	360	71987	57253	595	0	5.812	1	0.595	0	0.259	0	0.262
rbg378a	380	79678	64342	559	36	19.047	597	77.582	12	3.342	223	32.326
kro124p.1	101	9868	232	32597	0	1.782	2	0.064	2	0.064	2	0.062
kro124p.2	101	9833	267	32851	27	3.281	151	5.268	224	8.884	137	5.440
textbfkro124p.3	101	9635	465	33779	98	7.469	2000	308.242	2000	294.649	2000	279.382
kro124p.4	101	7596	2504	37124	0	1.672	16	0.669	66	3.399	12	0.531
p43.1	44	1796	96	2720	128	1.765	0	0.002	0	0.003	0	0.003
p43.2	44	1773	119	2720	237	4.359	0	0.002	0	0.004	0	0.039
p43.3	44	1711	181	2720	134	1.437	1	0.056	1	0.066	1	0.065
p43.4	44	1311	581	2820	353	2.797	2	0.041	0	0.011	0	0.009
prob.100	100	9662	238	650	4	743.969	125	5.565	48	2.377	37	1.560
prob.42	42	1622	100	143	0	0.032	0	0.002	0	0.002	0	0.007
ry48p.1	49	2245	107	13095	54	0.609	132	1.289	159	1.386	107	1.073
ry48p.2	49	2231	121	13103	0	0.235	15	0.197	13	0.138	10	0.099
textbfry48p.3	49	2125	227	13886	146	2.156	2000	24.712	2000	20.654	1961	18.655
ry48p.4	49	1661	691	15340	32	0.313	77	0.664	126	1.049	43	0.409
Average					45	20.855	132	10.616	121	8.372	114	8.420

between the tightness of the lower bound and computation time, and so that the number of multipliers does not grow too large.

The rate of increase of the number of the Lagrangian multipliers at the root node using different step size rules can be seen in Fig. 6. For the majority of the instances, preliminary experiments clearly suggested an increase in the solution time (sometimes considerable) and a slight reduction in the number of generated nodes. This leads to the conclusion that setting $M = 10$ provides a good balance between computation time and performance.

5.2. Overall results

The results of the B&B algorithm with different step size rules are presented and compared with the current state-of-the-art results reported in Chou et al. (2023), that are obtained by solving the MILP model introduced in Section 2. The results of the methods discussed in Dell'Amico et al. (2021) and Dell'Amico et al. (2023) are not compared against in this work as they are generally dominated with computation times that are 906.2% and 131.6% higher than those presented in Chou et al. (2023) on average.

The MILP model from Chou et al. (2023) is solved using CPLEX 12.8.[1] CPLEX is run with its default parameters, and single threaded standard Branch-and-Cut algorithm is applied for solving the model, with *BestBound* node selection, and MIP emphasis set to *MIPEmphasisOptimality*. The maximum number of iterations $M = 10$ is used for the B&B algorithm.

Tables 1–3 report the results for the three benchmark sets. The tables compare the performance of the current state-of-the-art solver described in Chou et al. (2023) with the B&B algorithm introduced in Section 4 using different step size rules. In all the tables, column *Name* reports the name of the instance, column $|V|$ reports the number of vertices in the instance graph, column $|A|$ reports the number of arcs in the instance graph, and column $|R|$ reports the number of precedence relationships for each instance. Column z^* reports the value of the optimal solution of each instance. For each computational method considered, we report the following columns: Column *Nodes* contains the number of nodes generated in the search-tree; Column *Time [s]* contains the solution time in second.

An overview of the results show that the B&B algorithm optimally solves the three benchmark sets. The MILP from Chou et al. (2023) has an average solution time of 27.5 s, while the B&B algorithm has an average solution time of 8.9 (a 67.6% decrease) seconds with $\alpha_k = 0.1$, an average solution time of 5.9 (a 78.5% decrease) seconds with $\alpha_k = \frac{1}{k}$, and an average solution time of 6.2 (a 77.5% decrease) seconds with $\alpha_k = \frac{1}{p}$. To solve the MILP from Chou et al. (2023) 77 search-tree nodes are generated on average, while the B&B algorithm generates 96 (a 24.7% increase) nodes on average with $\alpha_k = 0.1$, 74 (a 3.9% decrease) nodes with $\alpha_k = \frac{1}{k}$, and 73 (a 5.2% decrease) nodes with $\alpha_k = \frac{1}{p}$.

[1] IBM ILOG CPLEX Optimization Studio: https://www.ibm.com/products/ilog-cplex-optimization-studio.

Table 2
Overall computational results for comparing the MILP solver and the B&B algorithm for SOPLIB instances.

Instance					MILP (Chou et al., 2023)		B&B \w $\alpha_k = 0.1$		B&B \w $\alpha_k = \frac{1}{k}$		B&B \w $\alpha_k = \frac{1}{p}$	
Name	$\|V\|$	$\|A\|$	$\|R\|$	z^*	Nodes	Time [s]	Nodes	Time [s]	Nodes	Time [s]	Nodes	Time [s]
R.200.100.1	200	39403	397	29	0	0.875	0	0.040	0	0.031	0	0.030
R.200.100.15	200	9381	16858	454	177	64.812	1585	61.350	719	20.733	739	21.166
R.200.100.30	200	7989	19036	529	10	0.875	8	0.181	21	0.348	10	0.200
R.200.100.60	200	12547	19725	6018	0	0.094	0	0.049	0	0.048	0	0.056
R.200.1000.1	200	39403	397	887	0	0.656	0	0.031	0	0.028	0	0.029
R.200.1000.15	200	8644	17252	5891	87	7.860	50	1.006	71	1.267	29	0.653
R.200.1000.30	200	7941	19061	7653	0	0.297	7	0.178	13	0.230	6	0.147
R.200.1000.60	200	12450	19672	6666	0	0.094	0	0.064	0	0.046	0	0.051
R.300.100.1	300	89103	597	13	0	2.250	0	0.077	0	0.064	0	0.097
R.300.100.15	300	15619	40601	575	139	55.734	517	22.353	94	4.739	190	9.592
R.300.100.30	300	16446	43483	756	0	0.562	3	0.236	1	0.165	1	0.168
R.300.100.60	300	27864	44569	708	0	0.375	1	0.196	1	0.184	0	0.159
R.300.1000.1	300	89103	597	715	0	2.515	0	0.067	0	0.062	0	0.062
R.300.1000.15	300	15750	40589	6660	73	16.531	13	0.695	52	2.366	9	0.474
R.300.1000.30	300	16970	43290	8693	0	0.453	14	0.648	10	0.299	3	0.205
R.300.1000.60	300	27629	44559	7678	0	0.297	2	0.249	2	0.193	2	0.211
R.400.100.1	400	158803	797	6	2	9.750	0	0.141	0	0.122	0	0.115
textbfR.400.100.15	400	24381	73961	699	109	44.922	2000	425.474	2000	268.596	2000	291.932
R.400.100.30	400	28156	78076	712	0	2.031	2	0.481	3	0.397	5	0.633
R.400.100.60	400	49022	79452	557	0	0.328	0	0.379	0	0.285	0	0.322
R.400.1000.1	400	158803	797	780	0	2.797	0	0.130	0	0.106	0	0.119
R.400.1000.15	400	23785	74218	7382	91	24.000	702	41.821	199	10.402	168	9.102
R.400.1000.30	400	28074	77989	9368	38	6.563	27	2.376	51	4.411	31	2.510
R.400.1000.60	400	49155	79405	7167	0	0.500	0	0.351	0	0.354	0	0.348
R.500.100.1	500	248503	997	3	0	11.812	0	0.194	0	0.179	0	0.189
R.500.100.15	500	33522	117854	860	38	21.156	31	2.873	15	1.442	5	0.896
R.500.100.30	500	42873	122264	710	15	3.562	7	1.278	11	1.714	13	1.726
R.500.100.60	500	76419	124257	566	0	0.844	0	0.685	0	0.639	0	0.632
R.500.1000.1	500	248503	997	297	0	4.469	0	0.214	0	0.177	0	0.192
R.500.1000.15	500	34519	117314	8063	0	15.063	5	1.256	6	0.860	6	0.823
R.500.1000.30	500	42790	122387	9409	0	3.125	9	1.217	17	1.749	6	0.891
R.500.1000.60	500	76231	124270	6163	0	0.875	0	0.773	0	0.613	0	0.632
R.600.100.1	600	358203	1197	1	0	733.375	0	0.314	0	0.248	0	0.303
R.600.100.15	600	45825	170772	568	0	5.312	0	1.302	1	0.973	0	0.983
R.600.100.30	600	59865	177036	776	0	2.375	5	2.080	3	1.207	2	1.132
R.600.100.60	600	109237	179207	538	0	0.906	0	0.225	0	0.981	0	1.076
R.600.1000.1	600	358203	1197	322	0	8.625	0	0.303	0	0.260	0	0.299
R.600.1000.15	600	47457	169898	9763	0	12.766	4	1.155	5	1.238	1	1.143
R.600.1000.30	600	60330	176864	9497	0	2.969	3	1.184	7	1.375	2	1.228
R.600.1000.60	600	109595	179104	6915	0	0.922	0	0.336	0	0.987	0	1.089
R.700.100.1	700	487903	1397	2	0	314.875	0	0.446	0	0.336	0	0.380
R.700.100.15	700	58858	234155	675	0	6.875	8	2.587	1	1.544	2	1.734
R.700.100.30	700	80436	241484	590	0	1.250	0	0.975	0	0.973	0	0.961
R.700.100.60	700	148401	243968	383	0	1.422	0	0.994	0	0.982	0	0.991
R.700.1000.1	700	487903	1397	611	0	13.891	0	0.440	0	0.358	0	0.402
R.700.1000.15	700	59223	233939	2792	0	1.875	0	0.825	0	0.889	0	0.881
R.700.1000.30	700	81226	241265	2658	0	0.828	0	0.926	0	0.952	0	0.911
R.700.1000.60	700	148861	243953	1913	0	1.375	0	0.935	0	0.962	0	0.952
Average					16	29.494	104	12.127	69	7.023	67	7.476

The bypass rule described in Section 4.4 is invoked for three outlier instances (highlighted in the tables): *kro124p.3*, *ry48p.3*, and *R.400.100.15*, once the size of the search tree grows larger than 2000 nodes. Without applying the bypass rule on these instances, and enforcing a time limit of 1 h on the computation time, the results for the three instances were as follows. The instance *kro124p.3* times out with an optimality gap of 0.951% with $\alpha_k = 0.1$, 3.125% with $\alpha_k = \frac{1}{k}$, and 3.029% $\alpha_k = \frac{1}{p}$. Instance *ry48p.3* is solved to optimality with an average (among the B&B methods) solution time of 23.014 s, and 2530 nodes are generated on average. The instance *R.400.100.15* is solved optimally with $\alpha_k = \frac{1}{k}$ and $\alpha_k = \frac{1}{p}$, with an average solution time of 300.396 s, and 4542 nodes are generated on average. With $\alpha_k = 0.1$ the B&B times out on the instances with an optimality gap of 0.855%.

Excluding from the statistics the three outlier instances where the bypass rule is invoked, solving the MILP from Chou et al. (2023) takes on average 27.7 s, while the B&B algorithm has an average solution time of 2.4 (a 91.3% decrease) seconds with $\alpha_k = 0.1$, an average solution time of 0.9 (a 96.8% decrease) seconds with $\alpha_k = \frac{1}{k}$, and an average solution time of 1.1 (a 96.0% decrease) seconds with $\alpha_k = \frac{1}{p}$. In terms of the number of nodes generated in the search-tree, solving the MILP from Chou et al. (2023) requires 75 nodes on average, while the B&B algorithm generates 45 (a 40.0% decrease) nodes on average with $\alpha_k = 0.1$, 23 (a 69.3% decrease) nodes with $\alpha_k = \frac{1}{k}$, and 22 (a 70.7% decrease) nodes with $\alpha_k = \frac{1}{p}$. In summary, when excluding the three outlier instances, the B&B algorithm is on average 94.7% faster, with 60% less nodes generated in the search-tree.

In conclusion, the results show that using a B&B algorithm that is based on a Lagrangian relaxation of the problem is generally faster than current state-of-the-art methods, except on a very small subset of the instances.

6. Conclusions

This work introduces a branch-and-bound algorithm that is based on a Lagrangian relaxation for the Precedence-Constrained Minimum-Cost Arborescence problem.

The experimental results show that the newly proposed algorithm is 74.6% faster on average at solving standard benchmark instances compared to state-of-the-art methods currently available.

Table 3
Overall computational results for comparing the MILP solver and the B&B algorithm for COMPILERS instances.

Instance					MILP (Chou et al., 2023)		B&B \w $\alpha_k = 0.1$		B&B \w $\alpha_k = \frac{1}{k}$		B&B \w $\alpha_k = \frac{1}{p}$	
Name	$\|V\|$	$\|A\|$	$\|R\|$	z^*	Nodes	Time [s]	Nodes	Time [s]	Nodes	Time [s]	Nodes	Time [s]
gsm.153.124	126	1165	7611	185	3	0.140	0	0.019	0	0.012	0	0.016
gsm.444.350	353	1851	61612	1542	0	0.094	0	0.246	0	0.204	0	0.198
gsm.462.77	79	1365	2591	292	0	0.031	0	0.037	1	0.052	0	0.039
jpeg.1483.25	27	471	170	71	4	0.047	0	0.042	0	0.007	0	0.003
jpeg.3184.107	109	1845	5220	411	0	0.093	0	0.057	1	0.036	4	0.040
jpeg.3195.85	87	2359	2769	13	5674	897.312	9	2.992	6	2.940	6	3.062
jpeg.3198.93	95	2660	3357	140	401	9.704	27	0.334	68	0.551	162	2.742
jpeg.3203.135	137	2549	8352	507	7	0.125	6	0.075	1	0.028	0	0.022
jpeg.3740.15	17	237	35	33	0	0.031	33	0.099	31	0.098	5	0.019
jpeg.4154.36	38	651	445	74	0	0.063	0	0.014	4	0.036	65	0.526
jpeg.4753.54	56	940	1184	146	6	0.109	0	0.010	0	0.011	0	0.004
susan.248.197	199	3232	18495	588	0	0.125	1	0.063	0	0.048	0	0.053
susan.260.158	160	2795	11649	472	0	0.141	4	0.142	8	0.178	7	0.137
susan.343.182	184	2923	15759	468	19	0.359	12	0.355	8	0.203	13	0.365
typeset.10192.123	125	4558	5767	241	0	0.500	42	0.909	103	1.371	96	1.159
typeset.10835.26	28	573	132	60	0	0.031	0	0.002	0	0.001	0	0.001
typeset.12395.43	45	1130	513	125	0	0.078	0	0.004	0	0.003	0	0.003
typeset.15087.23	25	346	167	89	0	0.047	4	0.013	3	0.022	2	0.012
typeset.15577.36	38	783	390	93	0	0.015	0	0.002	0	0.002	0	0.017
typeset.16000.68	70	1990	1588	67	144	7.172	424	5.393	3	1.123	9	0.151
typeset.1723.25	27	603	86	54	21	0.110	127	0.888	117	0.558	106	0.528
typeset.19972.246	248	1797	30419	979	0	0.062	0	0.093	0	0.078	0	0.083
typeset.4391.240	242	2463	28620	837	0	0.094	2	0.105	1	0.083	1	0.087
typeset.4597.45	47	1268	533	133	0	0.031	0	0.009	0	0.051	0	0.004
typeset.4724.433	435	3354	93961	1819	0	0.172	0	0.038	0	0.372	0	0.405
typeset.5797.33	35	422	445	93	0	0.032	0	0.037	0	0.003	0	0.009
typeset.5881.246	248	2247	30187	979	0	0.343	14	0.255	5	0.133	19	0.295
Average					233	33.965	26	0.453	13	0.304	18	0.370

Future work to enhance the performance of the B&B algorithm will be related to the use of different subgradient methods (e.g. *Projected Subgradient methods*), and different step size rules. Moreover, different pruning and branching techniques can be devised for the evolution of the search-tree. Finally, different formulations of the problem could be considered.

CRediT authorship contribution statement

Mauro Dell'Amico: Conceptualization, Methodology, Writing – review & editing. **Jafar Jamal:** Conceptualization, Methodology, Software, Formal analysis, Investigation, Data curation, Writing – original draft, Visualization. **Roberto Montemanni:** Conceptualization, Methodology, Formal analysis, Investigation, Writing – review & editing, Supervision.

Data availability

Data will be made available on request.

References

Bazaraa, M.S., Sherali, H.D., 1981. On the choice of step size in subgradient optimization. European J. Oper. Res. 7 (4), 380–388.

Cai, M.C., Deng, X., Wang, L., 2004. Minimum k arborescence with bandwidth constraints. Algorithmica 38 (4), 529–537.

Chou, X., Dell'Amico, M., Jamal, J., Montemanni, R., 2023. Precedence-constrained arborescences. European J. Oper. Res. 307 (2), 575–589.

Chu, Y.J., Liu, T., 1965. On the shortest arborescence of a directed graph. Sci. Sin. 14, 1396–1400.

Dell'Amico, M., Jamal, J., Montemanni, R., 2021. A mixed integer linear program for a precedence-constrained minimum-cost arborescence problem. In: Proc. of the 8^{th} International Conference on Industrial Engineering and Applications (Europe). ICIEA-EU, Association for Computing Machinery, New York, NY, USA, pp. 216–221.

Dell'Amico, M., Jamal, J., Montemanni, R., 2023. Compact models for the precedence-constrained arborescences. In: Advances in Transdisciplinary Engineering 34 – Proc. of the 6^{th} International Conference on Intelligent Traffic and Transportation. ICITT, IOS Press, pp. 112–126.

Edmonds, J., 1967. Optimum branchings. J. Res. Natl. Bureau Stand. B 71 (4), 233–240.

Escudero, L.F., 1988. An inexact algorithm for the sequential ordering problem. European J. Oper. Res. 37 (2), 236–249.

Escudero, L.F., Guignard, M., Malik, K., 1994. A Lagrangian relax-and-cut approach for the sequential ordering problem with precedence relationships. Ann. Oper. Res. 50 (1), 219–237.

Fertin, G., Fradin, J., Jean, G., 2017. Algorithmic aspects of the maximum colorful arborescence problem. In: Theory and Applications of Models of Computation. TAMC 2017, Springer, pp. 216–230.

Fischetti, M., Vigo, D., 1997. A branch-and-cut algorithm for the resource-constrained minimum-weight arborescence problem. Networks: Int. J. 29 (1), 55–67.

Fisher, M.L., 1981. The Lagrangian relaxation method for solving integer programming problems. Manage. Sci. 27 (1), 1–18.

Frieze, A.M., Tkocz, T., 2021. A randomly weighted minimum arborescence with a random cost constraint. Math. Oper. Res..

Gabow, H.N., Galil, Z., Spencer, T., Tarjan, R.E., 1986. Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. Combinatorica 6 (2), 109–122.

Gouveia, L., Lopes, M.J., 2005. The capacitated minimum spanning tree problem: On improved multistar constraints. European J. Oper. Res. 160 (1), 47–62.

Houndji, V.R., Schaus, P., Hounkonnou, M.N., Wolsey, L., 2017. The weighted arborescence constraint. In: International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. Springer, pp. 185–201.

Li, J., Liu, X., Lichen, J., 2017. The constrained arborescence augmentation problem in digraphs. In: 2017 3rd IEEE International Conference on Computer and Communications. ICCC, IEEE, pp. 1204–1209.

Lucena, A., 1992. Steiner problem in graphs: Lagrangian relaxation and cutting planes. COAL Bull. 21, 2–8.

Montemanni, R., Smith, D.H., Gambardella, L.M., 2008. A heuristic manipulation technique for the sequential ordering problem. Comput. Oper. Res. 35 (12), 3931–3944.

Morais, V., Gendron, B., Mateus, G.R., 2019. The p-arborescence star problem: Formulations and exact solution approaches. Comput. Oper. Res. 102, 91–101.

Reinelt, G., 1991. TSPLIB–A travelling salesman problem library. ORSA J. Comput. 3 (4), 376–384.

Shobaki, G., Jamal, J., 2015. An exact algorithm for the sequential ordeing problem and its application to switching energy minimization in compilers. Comput. Optim. Appl. 61 (2), 343–372.

Toth, P., Vigo, D., 1995. An exact algorithm for the capacitated shortest spanning arborescence. Ann. Oper. Res. 61 (1), 121–141.