University of Modena and Reggio Emilia

Department of
Sciences and Methods for Engineering

PhD Program
INDUSTRIAL INNOVATION ENGINEERING

Precedence-Constrained Minimum Arborescence Problems

Supervisor:
Prof. Roberto Montemanni
Coordinator:
Prof. Franco Zambonelli

Candidate:
Jafar Jamal
Mat. 148360

Cycle XXXV

Abstract

The *Minimum-Cost Arborescence problem* is a well-studied problem in the area of graph theory, with known polynomial algorithms for solving the problem. Since the problem was introduced, various variations on the problem with different objective function and/or constraints were introduced in the literature.

In this work we introduce a new variation of the problem called the *Precedence-Constrained Minimum-Cost Arborescence problem*, in which precedence constraints are enforced on pairs of vertices in the graph. The purpose of the precedence constraints is to prevent the formation of directed paths in the arborescence that violate a precedence relationship. A precedence relationship enforced on the pair of vertices (s, t) imply that a directed path that leads from t to s could not appear in any feasible solution. We show that the Precedence-Constrained Minimum-Cost Arborescence problem is $\mathcal{NP}$-hard, and propose a set of mixed integer linear programming models for formulating the problem, and a branch-and-bound algorithm that is based on Lagrangian Relaxations. An experimental study is conducted which compares the performance of the models and the branch-and-bound algorithm.

An extension to the Precedence-Constrained Minimum-Cost Arborescence problem named the *Precedence-Constrained Minimum-Cost Arborescence problem with Waiting-Times* is also introduced in this thesis, that is characterized by an additional constraint that can be described as follows. Given a weighted directed graph with arc costs indicating the time required to traverse that arc, and assuming that

there is a flow that start at the root and traverses each path of the arborescence. The additional constraint imposes that for any precedence relationship (s, t) the flow must enter t at the same time or after entering s. The constraint implies that there is a waiting time at vertex t, if vertex t is closer to the root compared to vertex s. The objective of the problem is to find an arborescence that has a minimum total cost, plus total waiting times. We show that the Precedence-Constrained Minimum-Cost Arborescence problem with Waiting-Times is also $\mathcal{NP}$-hard, and propose a set of mixed integer linear programming models and constraint programming models for describing the problem. An experimental study is conducted which compares the performance of the models.

Acknowledgements

I am deeply grateful and thankful to my supervisor Prof. Roberto Montemanni, and Prof. Mauro Dell'Amico who generously provided help and guidance during the process of this thesis, and throughout my studies. I would also like to thank Prof. Massimo Paolucci from Università di Genova and Prof. Stefano Gualandi from Università di Pavia for their feedback and comments that helped improve the presentation of the material. I would also like to thank my dissertation committee (in alphabetical order): Prof. Carlo Concari from Università di Parma, Prof. Anna Maria Ferrari from Università di Modena e Reggio Emilia, Prof. Luca Maria Gambardella from Università della Svizzera italiana, Prof. Luca Montorsi from Università di Modena e Reggio Emilia, Prof. Lucia Pallottino from Università di Pisa, and Prof. Michele Pinelli from Università di Ferrara. Last but not least I am thankful to my friends and family for their support and encouragement.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

A major problem in the area of *distribution network design*, is determining the best way to transfer goods from a supply point to multiple points of demand by choosing the layout of the network, while minimizing the overall costs. A real-world example of such a network is a natural gas distribution network, which often relies on the use of pipelines in order to transfer the natural gas to a set of destination points. Therefore, choosing the best way to connect the destination points to the source point greatly impacts the operating and maintenance costs of the network, and consequently gas prices. In such networks, a primary objective among others, is to decide the minimum total length of pipelines needed such that every destination point in the network is reachable from the source point, possibly through a set of different destination points. In the area of graph theory, the problem of finding the minimum total length to connect a set of points in a network to a source point is named the *Minimum-Cost Spanning-Tree problem* [87], or the *Minimum-Cost Arborescence problem* [26] in the case where the goods can travel in a single direction.

In some special cases, it is desired for the commodity (such as natural gas) not to pass through a transit country on its way to the destination country. Such a scenario can occur when the two countries have unstable political relations, or high

cost transit duties that are enforced on the destination or source country by the transit country, which in turn can considerably increase the commodity price. In such a case the objective is to decide the minimum total length of pipelines needed such that every destination point in the network is reachable from the source point, and the commodity does not pass through a set of transit points on its way to a set of destination points. A *precedence-constraint* is constraint which imposes that the commodity must not pass through point a on its way to point b. The problem of finding the minimum total length of pipelines required to connect a set of destination points in a network to a source point, such that the precedence-constraints are respected between a set of points is named the *Precedence-Constrained Minimum-Cost Arborescence problem*. In other cases, it might be desired to define a priority level for the destination points motivated by contract clauses, such that the commodity reaches points with a lower priority once or after the commodity has reached all points with higher priority while satisfying the precedence-constraints. This transforms the problem to a *Precedence-Constrained Minimum-Cost Arborescence problem with Waiting-Times*.

In this thesis, we study the *Precedence-Constrained Minimum-Cost Arborescence problem* and the *Precedence-Constrained Minimum-Cost Arborescence problem with Waiting-Times*, and investigates the computational complexity of both problems. In addition, we use several combinatorial optimization techniques to solve the problems, and experimentally evaluate the performance of the several techniques used.

The rest of this chapter is organized as follows. Section 1.1 covers fundamental concepts of combinatorial and integer optimization. Section 1.2 covers basics concepts and terminology from graph theory that are used in this work. Finally, we end the chapter with an outline of the thesis. For a general overview of combinatorial optimization techniques, we refer the reader to the following books [72, 77, 88].

1.1 Combinatorial and Integer Optimization

Combinatorial optimization can be defined as the process of finding one or more optimal solutions in a well defined discrete problem space containing a finite set of possible solutions, that minimizes or maximizes a function called the *objective function*. The set of possible solutions can be defined by inequality and/or equality constraints, and integrality constraints that force the decision variables to be integers. The set of points that satisfy the defined constraints is called the *feasible solution set*.

Combinatorial optimization problems arise in some fields of management and logistics, such as production planning, scheduling, supply chain, facility location, and commodity transportation networks, as well as in some fields of engineering, such as design of bridges, and data networks.

Integer programming models often refer to combinatorial optimization models where most or all the decision variables can be assigned a finite number of possibilities. In this thesis we consider combinatorial optimization problems where the objective function and constraints are linear and the variables are integers or continuous. These problems are called *integer linear programming problems* and are defined as follows.

$$\begin{aligned} &\text{minimize } c^T x \\ &\text{subject to } Ax \leq b \\ &x \in \mathbb{Z}^n \end{aligned}$$

where $\mathbb{Z}^n$ is the set of integral n-dimensional vectors, and both x and c are integer n-vectors. Furthermore, A is an $m \times n$ matrix and b is an m-vector, where m is

number of inequality constraints. If the model contains another set of variables y that are continuous (i.e., $y \in \mathbb{R}^n$), then we obtain what is called a *mixed integer linear programming model* (MILP). In chapter 3 and chapter 4 we present several MILP models for the Precedence-Constrained Minimum-Cost Arborescence problem, and the Precedence-Constrained Minimum-Cost Arborescence problem with Waiting-Times.

Solving integer linear programming problems can sometimes be a difficult task, which arises from the fact that unlike linear problems where the feasible region is a convex set, in integer problems a lattice of feasible points must be searched, or in mixed integer problems a set of disjoint line segments must be searched to find an optimal solution. Therefore, in integer programming we may have many local optima and finding a global optimum to the problem requires a proof that a particular solution dominates all feasible points.

When solving combinatorial problems there is always a trade-off between the quality of the solution and the running time. We can either try to find the optimal solution to the problem with an *exact algorithm*, or find a sub-optimal solution using a *heuristic algorithm* which usually has a considerably smaller running time.

1.1.1 Fundamental Concepts of Complexity Theory

Complexity theory [15, 60, 66] can be described as the study of what is hard or easy to solve by a computer. Problem classification is dealt with using *complexity theory*, that is to determine the number of steps it takes to solve a problem relative to the input size. In this subsection we summarize the most important concepts of complexity theory.

An *algorithm* is defined as a list of steps or instructions that solve every instance of a specific problem in a finite number of steps. This also implies that an algorithm

can also detect if the problem instance has no solution.

The *size of a problem* is the amount of information needed to describe the instance, that is encoded by a string of symbols. Thus, the size of the instance is equal to the number of symbols in the string.

The *running time* of an algorithm is measured by an upper bound on the number of elementary arithmetic operations (addition, subtraction, multiplication, division, and comparison) executed on the input, expressed as a function of the input size. The input is the data used to encode a problem instance. Assuming that the input size is measured by n, then the running time of the algorithm is expressed as $O(f(n))$, if there exist a constant c and n_0 such that the number of steps for any instance with $n \geq n_0$ is bounded from above by $cf(n)$. An algorithm is said to be a *polynomial time algorithm* when its running time is bounded by a polynomial function $f(n)$, and an algorithm is said to be an *exponential time algorithm* when its running time is bounded by an exponential function such as $O(2^{f(n)})$.

Complexity theory is mainly concerned with *decision problems*, that is a problem that can be answered by either "*yes*" or "*no*". In the case of an integer programming problem the decision problem is described as follows. Given an instance of an integer programming problem and an integer k, is there a feasible solution x such that $c^T x \leq k$? The decision problems that are "*easy*" to solve are solvable in polynomial time and belong to the complexity class denoted by $\mathcal{P}$. This class of problems includes problems such as: finding the maximum number in a list of numbers, sorting a list of numbers, and solving linear programs.

On the other hand, the decision problems that are "*hard*" to solve are solvable in exponential time and belong to the complexity class EXP, where the majority of combinatorial problems belong. In order to distinguish between "*easy*" and "*hard*" problems we must first describe a complexity class that contains $\mathcal{P}$.

The complexity class $\mathcal{NP}$ is defined as the set of decision problems for which

the problem instances has a positive answer that can be verified in polynomial time by a *deterministic algorithm*, and the problem can be solved in polynomial time by a *non-deterministic algorithm*. The complexity class $\mathcal{NP}$-*complete* contains the hardest problems to solve that belong to $\mathcal{NP}$.

Definition 1.1. Polynomial transformation is an algorithm that for every instance α of problem Π_1 produces in polynomial time an instance β of problem Π_2, such that for every instance α of problem Π_1 the answer is "yes" if and only if the answer to instance β of problem Π_2 is "yes".

Definition 1.2. *A decision problem* Π *is classified as* $\mathcal{NP}$*-complete if* Π *in in* $\mathcal{NP}$ *and every other* $\mathcal{NP}$ *decision problem can be polynomially transformed into* Π.

Based on the previous definitions, if an $\mathcal{NP}$-*complete* problem can be solved in polynomial time, then all the problems that are in $\mathcal{NP}$ can be solved in polynomial time, and hence $\mathcal{P} = \mathcal{NP}$. Note that polynomial transformation is a transitive relation, that is if problem Π_1 is pollynomialy transferable to problem Π_2, and problem Π_2 is pollynomialy transferable to problem Π_3, then problem Π_1 is pollynomialy transferable to problem Π_3. Therefore if we want to prove that a decision problem Π is $\mathcal{NP}$-*complete* then we need to show that:

1. $\Pi \in \mathcal{NP}$.
2. Some decision problem that is already known to be $\mathcal{NP}$-*complete* can be polynomially transformed to Π.

Definition 1.3. *An* ***optimization problem*** Π *is classified as* $\mathcal{NP}$*-hard if there exists an* $\mathcal{NP}$*-complete decision problem that can be polynomially reduced to* Π.

An optimization problem is $\mathcal{NP}$-*hard* if the corresponding decision problem is $\mathcal{NP}$-*complete*. In particular, the two problems introduced in this thesis are $\mathcal{NP}$-*hard* (see section 3.3 and 4.2).

1.1.2 Heuristic and Relaxation Methods

In the previous section we have shown that when a combinatorial problem is shown to be $\mathcal{NP}$-*hard*, then that means a polynomial algorithm which solves the problem unlikely exists. However, some techniques can be used to find good solutions for hard optimization problems which requires the consideration of two issues: 1) calculating a *lower bound* that is close to the optimal solution as possible, and 2) calculating an *upper bound* that is as close to the optimal solution as possible.

Generating a good upper bound can be achieved by using general techniques (*Metaheuristics*) such as *Genetic/Evolutionary Algorithms* [18, 62], *Simulated Annealing* [24], *Tabu Search* [41], improvement heuristic such as *Local Search* [1], and problem specific heuristics. A key distinction between heuristics and metaheuristics, is that heuristics are designed to solve a particular problem, while the same metaheuristic algorithm can be used to solve multiple problems by tuning its inputs. Heuristics and Metaheuristics work well for combinatorial optimization problems, although they are not usually guaranteed to find a global optimal solution, however they can often find a sufficiently good solution very quickly, and are an alternative to exhaustive search, which would take an exponential amount of time relative to the size of the input. Furthermore, metaheuristics can be easily applied to a large number of problems. For example, in the case of genetic algorithms, all we need is a way to encode the possible solutions, but in principle, we can apply genetic algorithms to a wide range of problems, although they may not always be the best solution to each of these problems. Moreover, we can incorporate some randomness in order to escape local minima, which is usually done in genetic algorithms among others. For more information on metaheuristics the reader can refer to [42, 45, 82].

In this thesis, heuristic methods are not considered. However, every method we consider can be adapted to generate an upper bound by limiting the computa-

tion time and/or explored solution space of each method, which would considerably reduce the computation time of the solution method.

On the other hand, lower bounds can be computed using *relaxation* techniques. Relaxation is an approximation of a difficult problem by a nearby problem that is easier to solve, where a solution of the relaxed problem provides information about the original problem. Some of the techniques that are used to generate lower bounds that are used in this thesis are:

- **Linear Programming (LP) relaxation**

  Linear programming relaxation is a linear programming formulation that takes an integer programming formulation of a problem, and *relaxes* the integrality constraints on the set of integer variables, where the resulting relaxation is a linear program, hence the name. This relaxation technique transforms an $\mathcal{NP}$-hard optimization problem (integer programming) into a related problem that is solvable in polynomial time (linear programming). The linear program can be solved in polynomial time using standard algorithms such the *simplex method* [16] or *interior point method* [76]. The objective function value that is obtained by solving the linear programming model is a lower bound on the optimal objective function value to original minimization problem.

- **Lagrangian relaxation**

  Lagrangian relaxation methods [33] work by relaxing or dualizing a subset or all the constraints, by adding them to the objective function and associating each dualized constraint with a *Lagrangian multiplier*, that is used to penalize violations of equalities/inequalities, i.e. to impose a cost on the violations. These added costs are used instead of the strict inequality constraints in the optimization. In practice, the Lagrangian relaxation problem can often be solved more easily than the original problem, and the value achieved by solving

the Lagrangian relaxation is a lower bound on the original problem. The quality of the lower bound is mainly affected by the values chosen for the Lagrangian multipliers.

In more details, suppose we are given the integer linear programming model with $x \in \{0,1\}^n$, $A \in \mathbb{R}^{m,n}$, and $B \in \mathbb{R}^{m,n}$ of the following form:

$$\text{minimize } c^T x \tag{1.1}$$

$$\text{subject to } Ax \geq b_1 \tag{1.2}$$

$$Bx \geq b_2 \tag{1.3}$$

Now suppose that the problem can be solved in polynomial time by relaxing constraints (1.3). Based on that we may introduce constraint (1.3) into to objective function and associate the constraint with a set of nonnegative Lagrangian multipliers $\lambda = [\lambda_1, \lambda_2, \ldots, \lambda_m]$, which results in the following integer linear programming model, that is a Lagrangian relaxation of our original problem.

$$\text{minimize } c^T x + \lambda^T (b_2 - Bx) \tag{1.4}$$

$$\text{subject to } Ax \geq b_1 \tag{1.5}$$

The idea of the new model is that we get penalized if constraint (1.3) is violated, and rewarded if we satisfy the constraint strictly. In section 3.5 we will show how we can find appropriate values for the Lagrangian multipliers such that the solution of the relaxation is an optimal solution to the original

problem.

1.2 Fundamental Concepts of Graph Theory

In this section we summarize a set of terminology and definitions related to the problems introduced in this thesis.

Throughout the thesis we will be focusing only on directed graphs, which is the general form of graphs. A *directed graph* (*digraph*) is an ordered pair $G = (V, A)$, where $V = \{1, \ldots, n\}$ is a nonempty finite set of vertices, and $A \subseteq V \times V$ is a set of ordered pairs of vertices (i, j) with $i, j \in V$. The set A is called the arcs of the graph. If $(i, j) \in A$ then vertex i and vertex j are *adjacent vertices*. In such a case vertex j is called the *head* or *target* of (i, j), and vertex i is called the *tail* or *source* of (i, j). Two arcs are *adjacent* if they share the same source or tail. For example the two arcs (i, j) and (i, k) are adjacent since they both share the same source which is vertex i. A vertex j for which $(i, j) \in A$ is a neighbor of i, however i is not a neighbor of j if $(j, i) \notin A$. A *subgraph* $G' = (V', A')$ is a directed subgraph of G if $V' \subseteq V$ and $A' \subseteq A$, and $A' \subseteq V' \times V'$. A *directed weighted graph* is a directed graph where each arc $(i, j) \in A$ is associated with a weight c_{ij}, and the weight of an arc set $A' \subseteq A$ is given by $c(A') = \sum_{(i,j)\in A'} c_{ij}$.

We define the following notations to describe a specific set of arcs. Let $i \in V$ be a vertex, and let $U \subseteq V$ be a set of vertices.

$$\delta^+(i) = \{(i,j) \in A | j \in V\} \tag{1.6}$$

$$\delta^-(i) = \{(j,i) \in A | j \in V\} \tag{1.7}$$

$$\delta^+(U) = \{(i,j) \in A | i \in U, j \notin U\} \tag{1.8}$$

$$\delta^-(U) = \{(i,j) \in A | i \notin U, j \in U\} \tag{1.9}$$

The aforementioned sets of arcs can be described as follows. $\delta^+(i)$ is the set of arcs leaving vertex i, $\delta^-(i)$ is the set of arcs entering vertex i, $\delta^+(U)$ is the set of arcs leaving the set of vertices U, and $\delta^-(U)$ is the set of arcs entering the set of vertices U. Moreover, $A' \subseteq A$ is called an (s,t)-*cut* if $A' = \delta^+(U)$ such that $s \in U$ and $t \notin U$.

For a given directed graph $G = (V, A)$ with $s, t \in V$. A function $f : A \to \mathbb{R}$ is called an (s,t)-*flow* if:

1. $f(a) \geq 0 \ \forall a \in A$
2. $\sum_{a \in \delta^+(v)} f(a) = \sum_{a \in \delta^-(v)} f(a) \ \forall v \in V \backslash \{s,t\}$

Then by definition the value of an (s,t)-*flow* f is $c(f) = \sum_{a \in \delta^+(s)} f(a) - \sum_{a \in \delta^-(s)} f(a)$. This implies that the total amount of flow leaving s is equal to the total amount of flow entering t.

Given a directed graph $G = (V, A)$ and a capacity function $c : A \to \mathbb{R}_+$ that is associated with each arc $a \in A$, a flow f is said to be under c if $f(a) \leq c(a)$ for all $a \in A$. The *Maximum-Flow problem* is to find an (s,t)-*flow* under c that is of maximum value. The capacity of a cut $\delta^+(U)$ is defined as $c(\delta^+(U)) = \sum_{a \in \delta^+(U)} c(a)$. The value of a maximum-flow is equal to the capacity of a minimum-cut [34].

Theorem 1.1. Max-Flow Min-Cut [52] *For any directed graph* $G = (V, A)$,

$s, t \in V$, and $c : A \to \mathbb{R}_+$, the maximum value of an (s,t)-flow under c is equal to the minimum capacity of an (s,t)-cut. □

Given a directed graph $G = (V, A)$ then the set of arcs $P = \{(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k)\}, k \geq 2, v_i \in V, (v_k, v_{k+1}) \in A$ for all $i = 1, \ldots k$, is called a *walk* or a (v_1, v_k)-*walk*. A walk consisting of vertices that are all pairwise different, that is $v_i \neq v_j$ for all $i \neq j$, is called a *path* or a (v_1, v_k)-*path*. A walk $C = \{(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k), (v_k, v_1)\}$ that starts and ends at the same vertex is called a *cycle*. A cycle is called a *simple cycle* if all vertices are pairwise different, that is $v_i \neq v_j$ for all $i, j = 1, \ldots, k$, $i \neq j$. A directed graph $G = (V, A)$ is called an *acyclic graph*, if its arc set A does not contain a cycle.

Given a directed graph $G = (V, A)$, and $v_i, v_j \in V$, the two vertices v_i, v_j are said to be *connected*, if there exists an (i, j)-*path* in G (i.e., G contains a path that starts from vertex v_i and ends in vertex v_j).

Given a directed graph $G = (V, A)$, a *branching* B is an acyclic arc set such that every vertex in V is the target of at most one arc in B (i.e., each vertex has exactly one arc entering it). Furthermore, a connected branching is called an *arborescence*. For any arborescence there exists one vertex r called the *root* of the arborescence that is not the target of any arc in the branching (i.e., $\delta^-(r) = \phi$), from which there is a unique path from the root to every other vertex in the arborescence.

1.3 Outline of the Thesis

In this thesis we introduce the *Precedence-Constrained Minimum-Cost Arborescence problem* (PCMCA), and its variation, the *Precedence-Constrained Minimum-Cost Arborescence problem with Waiting-Times*. For both problems we introduced a proof of complexity, and we apply several techniques of combinatorial optimization to solve the problems. The rest of this thesis is organized as follows.

Chapter 2 describes the *Minimum-Cost Arborescence problem*, that is a special case of the *Precedence-Constrained Minimum-Cost Arborescence problem*, and we mention two efficient algorithm for constructing a minimum-cost arborescence, namely *Edmonds algorithm* [26] and a more efficient implementation of the algorithm proposed by Gabow and Tarjan [37]. Finally, we summarize variations of the *Minimum-Cost Arborescence problem* and other related work available in the literature.

Chapter 3 defines the *Precedence-Constrained Minimum-Cost Arborescence* (PCMCA) problem. In this chapter we present a proof of complexity for the problem, and propose several mixed integer linear programming formulation for the problem. We also propose a *Branch-and-Bound* algorithm for the PCMCA problem that uses the Lagrangian Relaxation technique discussed briefly in section 1.1.2, and introduce the principles of *Lagrangian relaxation* technique, and how to solve the Lagrangian relaxation. Finally, computational results and comparison between models and the different solving methods are presented.

Chapter 4 introduces the *Precedence-Constrained Minimum-Cost Arborescence problem with Waiting-Times*, where we present a proof of complexity for the problem, and propose several mixed integer linear programming, and constraint programming formulations for the problem. Computational results and comparisons between the models are presented. Finally, conclusions and summaries are drawn in chapter 5.

Chapter 2

The Minimum-Cost Arborescence Problem

The aim of this chapter is to provide the reader an overview of the classical *Minimum-Cost Arborescence* (MCA) problem [26]. In the first section we define the MCA problem. In the second section we formally present several MILP models from the literature. In the final section we present an overview of polynomial time algorithms for the MCA problem and their computational complexity.

2.1 Problem Definition

The MCA problem [26] can be describe as follows. Given a directed weighted graph and a *root* vertex, the objective of the problem is to find an arc subset of size $n-1$ that has a total minimum-cost, such that there are no arcs entering the root, and there is a *unique path* from the root to every other vertex in the graph. The problem where the root vertex is predefined is often called an *r-arborescence*. The problem has many practical applications in telecommunication and computer networks [64, 9], transportation problems [47], and tracking systems [51], etc. Furthermore, it can be

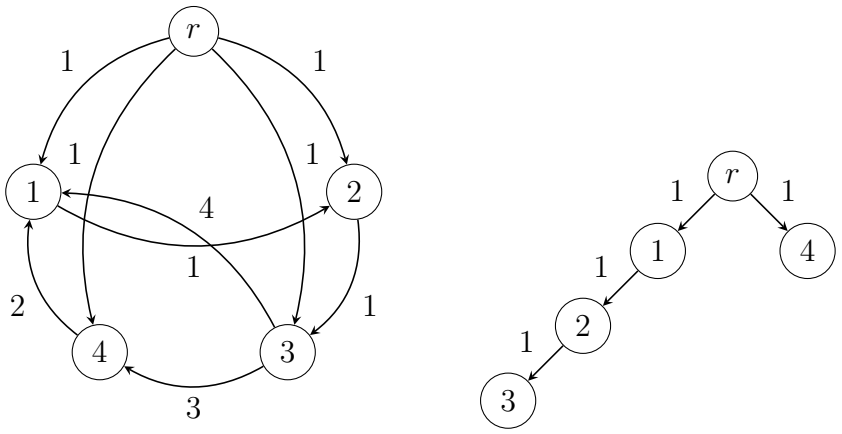

Figure 2.1: Example of a minimum-cost arborescence.

considered as a subproblem in many routing and scheduling problems [31].

As described in [26], the MCA problem can be defined on a directed graph $G = (V, A)$ where $V = \{1, \ldots, n\}$ is the set of vertices, and $A \subseteq V \times V$ is the set of arcs, where each arc $(i, j) \in A$ represents a connection between a pair of vertices $i, j \in V$. A cost c_{ij} is associated with every arc $(i, j) \in A$, and a root vertex $r \in V$ is normally provided. A feasible solution T is defined as a set of arcs (i.e. $T \subseteq A$) of size $|V| - 1$, that form a connected acyclic graph rooted at r. The objective function of the MCA can be reduced to the minimization of the following quantity $o(T)$, given a feasible subset of arcs T:

$$o(T) = \sum_{(i,j) \in T} c_{ij} \tag{2.1}$$

Figure 2.1 shows an example of a MCA. The graph on the left is the instance graph with its respective arc costs, while the graph on the right shows an optimal arborescence of cost 4. The arcs that are part of the arborescence is the set of arcs T.

2.2 Mixed Integer Linear Programming Models

In this section we present several MILP models for the MCA problem, that are either exponential or polynomial in size.

2.2.1 Set-Based Model

Let x_{ij} be a variable associated with every arc $(i,j) \in A$ such that $x_{ij} = 1$ if $x_{ij} \in T$ and 0 otherwise, and let $\delta^-(S)$ be the set of arcs entering the set of vertices S (see Section 1.2). The MCA problem can be formulated as the following binary integer linear program.

$$\text{minimize} \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{2.2}$$

$$\text{subject to} \sum_{(i,j)\in A} x_{ij} = 1 \qquad \forall j \in V\backslash\{r\} \tag{2.3}$$

$$\sum_{(i,j)\in \delta^-(S)} x_{ij} \geq 1 \qquad \forall S \subseteq V\backslash\{r\} \tag{2.4}$$

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

Constraints (2.3) enforces the first property of an arborescence that every vertex $v \in V\backslash\{r\}$ must have a single parent. Constraints (2.4) models the connectivity constraints, that is every vertex $v \in V\backslash\{r\}$ must be reachable from the root. Finally, constraints (2.5) define the domain of the variables. It should be noted that the set of constraints (2.3) can be omitted from the model [44]. The set of constraints (2.3) are left as part of the model as they affect the strength of the linear relaxation of the models proposed in the following chapters.

The *Set-Based* model uses an exponential number of connectivity constraints

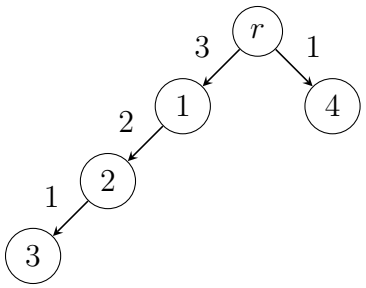

Figure 2.2: Example of a feasible MCA solution using a network flow based formulation.

(2.4), and is the basis of all polynomial time algorithms proposed for solving the MCA problem. Moreover, it is the most efficient model at solving the problem, since in practice, the number of constraints that are dynamically added to the model is small, and are added only when they are violated. On the other hand, the MCA problem can be modeled using a polynomial number of constraints, as shown in the next section.

2.2.2 Flow-Based Model

The MCA problem can alternatively be modeled using a polynomial number of constraints as a *network flow* problem [2]. Assuming that a flow of value at least one must reach every vertex in the graph other than the root, then if arc $(i, j) \in A$ is part of the solution (i.e. $(i, j) \in T$), then the amount of flow passing through the arc must be equal to the number of vertices reachable from vertex j plus one. The aforementioned constraints imply that the flow passing through arcs entering leaf vertices in the arborescence is equal to one, and the flow passing through arcs entering non-leaf vertices is greater than one. Since there are no arcs entering the root, and every vertex must be reachable from the root, then the sum of the flow leaving the root must be equal to $n-1$. Figure 2.2 shows an example of the amount of flow passing through every arc in a feasible solution. In the figure each arc $(i, j) \in T$ has a weight equal to the amount of flow passing through that arc.

Let y_{ij} be a variable associated with every arc $(i,j) \in A$ that is equal to the amount of flow passing through that arc, then the MCA problem can be formulated as the following integer linear programming model.

$$\text{minimize} \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{2.6}$$

$$\text{subject to} \sum_{(i,j)\in A} y_{ij} - \sum_{(j,i)\in A} y_{ji} = \begin{cases} -1 & i \neq r \\ n-1 & i = r \end{cases} \qquad \forall i \in V \tag{2.7}$$

$$x_{ij} \geq \frac{y_{ij}}{n-1} \qquad \forall (i,j) \in A \tag{2.8}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i.j) \in A \tag{2.9}$$

$$y_{ij} \in \mathbb{Z}^{+} \qquad \forall (i.j) \in A \tag{2.10}$$

Constraints (2.7) are the flow-connectivity constraints that enforce the following:

- The amount of flow entering the root is equal to zero, and hence no arcs enter the root.
- There is a unique path which connects every vertex in the graph to the root.
- Any feasible solution is a connected acyclic graph (i.e. an arborescence).

Constraints (2.8) impose the that the value of x_{ij} must be between zero and one if the value of y_{ij} is greater than zero. The value of y_{ij} is divided by $n-1$ since $y_{ij} \leq n-1$ from (2.7), which would restrict the value of x_{ij} to be between zero and one. Finally, constraints (2.9) and (2.10) define the domain of the variables.

2.3 Polynomial-Time Algorithms

In the related literature, the optimization problem of the MCA has been shown to be solvable in polynomial time. For further reading about the proof we refer the interested reader to [40, 43, 44].

The first polynomial time algorithm for solving the problem was proposed independently by Yoeng-Jin Chu and Tseng-Hong Liu [13], and Jack Edmonds [26]. A different polynomial time algorithm that operates directly on the cost matrix was discussed by Bock [7]. Both Edmond's and Chu-Liu algorithms have a computational complexity of $O(|E||V|)$. A faster implementation with a computational complexity of $O(|V| \log |V| + |A|)$, using *disjoint sets* [87] and a special implementation of *Fibonacci heaps* [35], was later on proposed by Gabow and Tarjan [37]. Edmond's algorithm is described in algorithm 1.

Algorithm 1 Edmond's algorithm for finding a MCA in G.

1: **procedure** FIND_MCA(G, r)
2: $\delta^-(r) = \phi$
3: $\pi(j) = null \, \forall j \in V$
4: Construct(G, r, π)
5: **end procedure**
6:
7: **procedure** CONSTRUCT(G, r, π)
8: **for** $j \in V \backslash \{r\}$ **do**
9: $\pi(j) =$ Get_Min(G, j)
10: **end for**
11: $P = \{(\pi(j), j | j \in V \backslash \{r\})\}$
12: $C =$ Contains_Cycle(P)
13: **if** $C \neq null$ **then**
14: Construct a directed graph $G' = (V', A')$ such that:
15: $V' = \{i \in V | i \notin C\} \cup \{v_C\}$
16: $A' = \{(i, j) | (i, j) \in A, (i, j) \notin C\}$
17: $c_{ij} = \begin{cases} c_{ij} - c_{\pi(j)j}, i \notin C \text{ and } j \in C \\ c_{ij}, i \in C \text{ and } j \notin C \\ c_{ij}, i \notin C \text{ and } j \in C \end{cases}$
18: Construct(G', r, π)
19: **end if**
20: **end procedure**

Line 1 removes all the arcs entering the root r. Line 2 creates an array which holds the parent of each vertex. Line 4 calls the function *Construct* which find a MCA in G rooted at r. Lines 8-10 assigns the parent of each vertex j as the source of the minimum weighted arc entering j. Line 11 creates the set P which is all the arcs currently selected as part of the arborescence. Line 12 checks if P contains a cycle and returns it. Lines 14-17 constructs a new graph G' by contracting the cycle C and adjusting the weights of the arcs as defined. Line 18 calls *Construct* on the contracted graph G'. Note that a MCA in G is found once the solution contains no cycles. The final MCA can be constructed by selecting the arcs in reverse order as

they were selected by the algorithm.

2.4 Literature Review

In this section we provide the reader with an overview of problems related to the MCA problem.

Since the MCA problem was first proposed, different variations have been introduced such as the *Resource-Constrained Minimum-Weight Arborescence* problem [32], where finite resources are associated with each vertex in an input graph. The objective of the problem is to find an arborescence with minimum total cost where the sum of the costs of outgoing arcs from each vertex is at most equal to the resource of that vertex. The problem is categorized as $\mathcal{NP}$-hard as it generalizes the *Knapsack* problem [32].

The *Minimum Spanning Tree* (MST) problem, is the problem of finding a minimum-cost tree which spans all the vertices of an undirected graph. The *Capacitated Minimum Spanning Tree* problem [47] is a variation of the MST problem, where a non-negative integer node demand q_j is associated with each vertex $j \in V\backslash\{r\}$, and an integer capacity Q is given. The objective is to find a minimum spanning tree such that the sum of demands of every subtree does not exceed a given capacity Q. The problem is shown to be $\mathcal{NP}$-hard as the particular case with zero cost arcs is a *bin packing* problem [47].

The *p-Arborescence Star* problem [74] is a relevant problem that is described as follows. Given a weighted directed graph $G = (V, A)$, a root vertex $r \in V$, and an integer p, the objective of the problem is to find a minimum-cost reverse arborescence rooted at r such that the arborescence spans the set of vertices $H \subseteq V\backslash\{r\}$ of size p, and each vertex $v \in V\backslash\{H \cup r\}$ is assigned to one of the vertices in H. The problem is shown to be $\mathcal{NP}$-hard [70] in the general case by a reduction from the *p-median*

problem [49].

Frieze and Tkocz [36] study the problem of finding a minimum-cost arborescence such that the cost of the arborescence is at most c_0. The problem is studied on randomly weighted digraphs where each arc in the graph has a weight w and a cost c, each being an independent uniform random variable U^s where $0 < s \leq 1$, and U is uniform $[0, 1]$. The problem is $\mathcal{NP}$-hard [36] through a reduction from the *Knapsack* problem.

Another problem of interest is the *Maximum Colorful Arborescence* problem [30] that can be described as follows. Given a weighted directed acyclic graph with each vertex having a specified color from a set of colors C, the objective is to find an arborescence of maximum weight, in which no color appears more than once. The problem is known to be $\mathcal{NP}$-hard [8] even when all arcs have a weight of 1.

The Constrained Arborescence Augmentation problem [63] is a different variation on the MCA problem that can be described as follows. Given a weighted directed graph $G = (V, A)$, and an arborescence $T = (V, A_r)$ in G rooted at vertex $r \in V$, the objective of the problem is to find an arc subset A' from $A - A_r$ such that there still exists an arborescence in the new graph $G' = (V, A_r \cup A' - a)$ for each arc $a \in A_r$, where the sum of the weights of the arcs in A' is minimized. The problem is an extension on the *Augmentation* problem [29], and is shown to be $\mathcal{NP}$-hard [63].

The *Minimum k Arborescence with Bandwidth Constraints* problem [9] is another variation, where every arc $a \in A$ has an integer bandwidth $b(a)$ that indicates the number of times such an arc can be used. The objective of the problem is to find k arborescences of minimum-cost rooted at the k given root vertices, covering every arc $a \in A$ at most $b(a)$ times. It has been shown that the problem can be solved in polynomial time [9].

The *Degree-Constrained Minimal Spanning Tree Problem with Unreliable Links and Node Outage Costs* [59] is modeled as a directed graph with the root vertex

being the central node of a network, and all other vertices being terminal nodes. The problem consists in finding links in a network to connect a set of terminal nodes to a central node, while minimizing both link costs and node outage costs. The node outage cost is the economic cost incurred by the network user whenever that node is disabled due to failure of a link. The problem is shown to be $\mathcal{NP}$-hard by reducing the problem to an equivalent *Traveling Salesman Problem* [39].

The *Minimum Changeover Cost Arborescence* problem [38] is another variation, where each arc is labeled with a color out of a set of k available colors. A changeover cost is defined on every vertex v in the arborescence other than the root. The cost over a vertex v is paid for each outgoing arc from v and depends on the color of its outgoing arcs, relative to the color of its incoming arc. The costs are given through a $k \times k$ matrix C, where each entry C_{ab}, specifies the cost to be paid at vertex v when its incoming arc is colored a and one of its outgoing arcs is colored b. A changeover cost at vertex v is calculated as the sum of costs paid for every outgoing arc at that vertex. The objective of the problem is to find an arborescence T with minimum total changeover cost for every vertex $j \in V$ other than the root. The problem is shown to be $\mathcal{NP}$-hard and very hard to approximate [38].

Finding a pair of arc-disjoint *in-arborescence* and *out-arborescence* is another problem, with the objective of finding a pair of arc-disjoint *r-out-arborescence* rooted at r_1 and *r-in-arborescence* rooted at r_2 where $r_1, r_2 \in V$. An *r-out-arborescence* has all its arcs directed away from the root, and an *r-in-arborescence* has all its arcs directed towards the root. A linear-time algorithm for solving the problem in directed acyclic graphs is proposed by Bérczi et al. [6]. The problem is shown to be $\mathcal{NP}$-complete in general graphs even if $r_1 = r_2$ [4].

Yingshu et al. [64] study the problem of constructing a strongly connected broadcast arborescence with bounded transmission delay. They devise a polynomial time algorithm for constructing a broadcast network with minimum energy consumption

that respects the transmission delays of the broadcast tree simultaneously.

The *Minimum Spanning Tree Problem with Conflict Pairs* is a variation of the minimum spanning tree problem where given an undirected graph and a set S that contains *conflicting pairs* of edges called a *conflict pair*, the objective of the problem is to find a minimum-cost spanning tree that contains at most one edge from each conflict pair in S [10]. The problem is shown to be $\mathcal{NP}$-hard [17].

The *Least-Dependency Constrained Spanning Tree* problem [86] is another variation that can be defined as follows. Given a connected graph $G = (V, E)$ and a directed graph $D = (E, A)$ whose vertices are the edges of G, the directed graph D is a *dependency graph* for E, and $e_1 \in E$ is a dependency of $e_2 \in E$ if $(e_1, e_2) \in A$. The objective of the problem is to decide whether there is a spanning tree T of G such that each edge in T has either an empty dependency or *at least* one of its dependencies is also in T. The *All-Dependency Constrained Spanning Tree* problem [86] is a similar problem that consists in deciding whether there is a spanning tree T of G such that each of its edges either has no dependency or all of its dependencies are in T. The two problems are shown to be $\mathcal{NP}$-complete [86].

2.5 Conclusions

In this chapter we defined the *Minimum-Cost Arborescence problem*, and presented several MILP models for the problem, that are exponential or polynomial in size. An overview of several polynomial time algorithms for solving the problem are mentioned. Finally, we have covered several variations on the MCA problem, and noticed that the majority of those variations are $\mathcal{NP}$-hard or $\mathcal{NP}$-complete problems.

Chapter 3

The Precedence-Constrained Minimum-Cost Arborescence Problem

The aim of this chapter is to introduce the *Precedence-Constrained Minimum-Cost Arborescence* (PCMCA) problem (first introduced in [19]), that is an extension to the MCA problem discussed in chapter 2. In the section 3.1 we define the PCMCA problem. Section 3.2 covers some applications of the MCA problem. Section 3.3 presents a proof of complexity for the PCMCA problem. In section 3.4 we propose several MILP models for the PCMCA problem, that use different techniques to model the precedence constraints. In section 3.5 we present *Lagrangian relaxation* based *Branch-and-Bound* algorithm for the PCMCA problem. Section 3.6 discusses computational results and experimentally compares the different models and methods proposed, while some conclusions are outlined in section 3.7. The work presented in this chapter has appeared in [12, 19, 21, 20].

3.1 Problem Definition

The PCMCA problem can be described by extending the definition of the MCA problem (see section 2.1). Given a directed weighted graph, a set R of ordered pairs of vertices, and a *root* vertex, the objective of the problem is to find a minimum-cost r-arborescence, such that for any $(s,t) \in R$ with $s,t \in V$, a feasible solution is an arborescence that does not contain a directed path from t to s. Alternatively, given a set R of ordered pairs of vertices, then for each precedence $(s,t) \in R$ any path of the arborescence covering both vertices s and t must visit s before visiting t.

The PCMCA problem can be formally defined on a directed graph $G = (V, A, R)$, where $V = \{1, \ldots, n\}$ is the set of vertices, $A \subseteq V \times V$ is the set of arcs, with each arc $(i,j) \in A$ representing a connection from a vertex $i \in V$ to a vertex $j \in V$, and $R \subset V \times V$ is the set of precedence relationships. The set of precedence relationships has to be acyclic, otherwise the problem instance does not contain a feasible solution. A cost c_{ij} is associated with every arc $(i,j) \in A$, and a root vertex $r \in V$ is provided. A feasible solution T is defined as a set of arcs (i.e. $T \subseteq A$) of size $|V| - 1$, that form a connected acyclic graph rooted at r, such that for each $(s,t) \in R$, t must not belong to the unique path that connects r to s. The objective function of the PCMCA can be reduced to the same objective function (2.1) for the MCA, given the additional constraints.

Figure 3.1 shows an example of a PCMCA. The graph on top is the instance graph with its respective arc costs, and the precedence relationship $(3,1) \in R$ is marked as a dashed arrow. The graph on the bottom left shows an optimal MCA of cost 4, while the graph on the bottom right shows an optimal PCMCA of cost 5. 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 a feasible PCMCA, then vertex 1 must succeed vertex 3 on the same directed path,

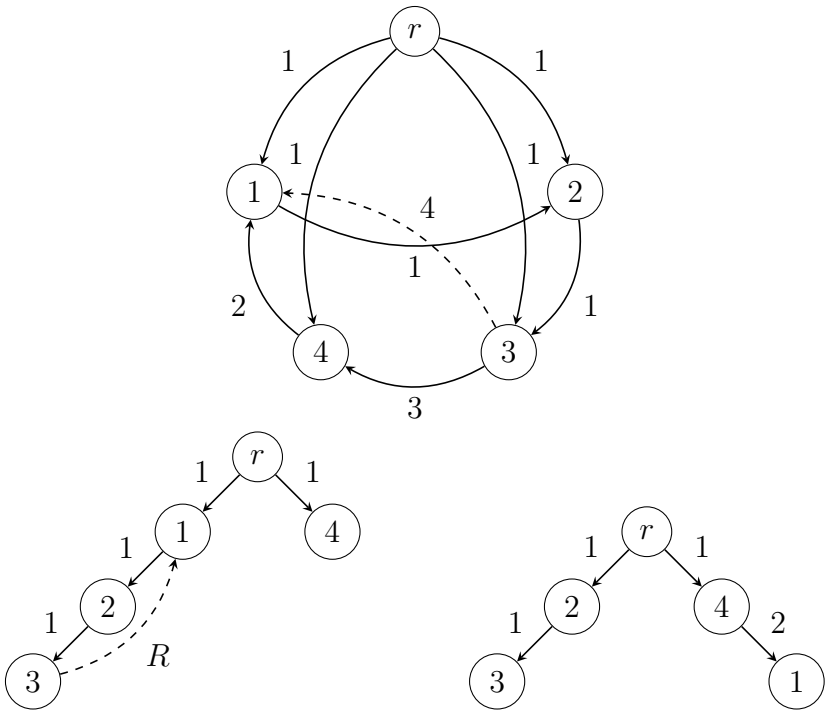

Figure 3.1: Example of a precedence-constrained minimum-cost arborescence.

or the two vertices must reside on two disjoint paths.

3.2 Applications

The PCMCA problem has practical applications in designing commodity distribution networks [19]. As an example, assume there is a main reservoir r that is filled with a commodity such as oil or gas, and we would like to connect r to a set of destination reservoirs (located in other countries/cities) in order to distribute the commodity, while at the same time minimizing the cost of the distribution infrastructure. The structure of the distribution network follows the definition of a MCA, where the main reservoir is the root vertex, and the destination reservoirs are the remaining vertices of the graph. The reason why the distribution is an arborescence, is because the flow always travels in a single direction away from the root (main reservoir). Now assume that a flow from r to a destination vertex t cannot pass

through another vertex s. Such a scenario can occur if two countries (s and t) have unstable political relations, therefore destination point t might not want to use destination point s as a transit point. In other cases, transit duties that are higher than the travel cost might be required to be paid in order to utilize that country's distribution infrastructure. Such a case occurred when Belarusian president Alexander Lukashenka has threatened to slap new duties on Russian oil transit unless Russian oil companies agree to sell Belarus oil at a lower price [81]. To avoid the aforementioned situations (and possibly others), then we can enforce a precedence-constraint on s and t (i.e., $(t, s) \in R$), and therefore the flow from r to t will not pass through s.

3.3 Computational Complexity

In this section we present a proof that the PCMCA problem is $\mathcal{NP}$-hard by a reduction from the 3-SAT problem [60]. The proof is inspired by the one introduced in [61] for the *Path Avoiding Forbidden Pairs problem*.

The 3-SAT problem can be described as follows. Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of variables. A *literal* in a boolean formula is an occurrence of a variable or its negation. A boolean formula is in conjunctive normal form (CNF), if its expressed as an AND ($\wedge$) of *clauses*, each of which is the OR ($\vee$) of one or more literals. A boolean formula is in *3-conjunctive normal form*, or 3-CNF, if each clause has exactly three distinct literals. For example, the boolean formula $(x_1 \vee \neg x_1 \vee x_2) \wedge (x_2 \vee x_3 \vee x_4)$ is in 3-CNF. In 3-SAT, we are asked whether a given boolean formula ϕ in 3-CNF is satisfiable. A boolean formula ϕ is satisfiable if there exists a truth assignment for ϕ which evaluates to *true*. The 3-SAT problem is known to be $\mathcal{NP}$-*complete* [60].

Theorem 3.1. *The PCMCA problem is $\mathcal{NP}$-hard*

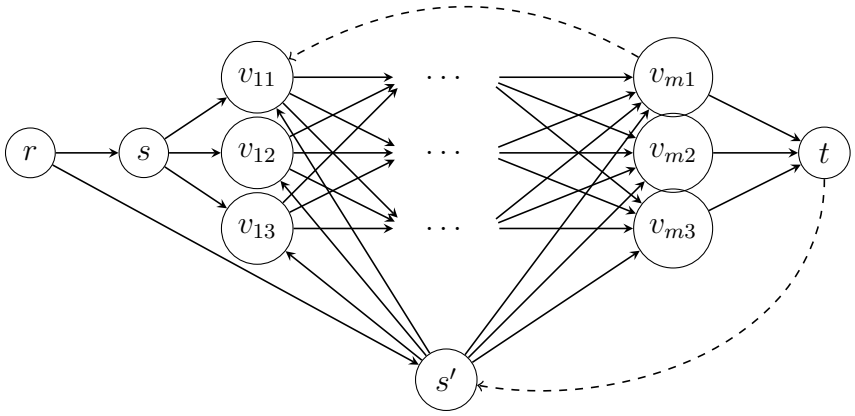

Figure 3.2: A PCMCA instance reduced from 3-SAT.

Proof. By reduction from 3-SAT: Let $X = \{x_1, x_2, \ldots, x_t\}$ be a set of variables. Let $\Phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ be a boolean expression in 3-conjunctive normal form, such that each clause C_i, $i = 1, \ldots, m$, is denoted by $(v_{i1} \vee v_{i2} \vee v_{i3})$, where each literal $v_{ik}, 1 \leq k \leq 3$, is associated to one variable in X or its negation. We will construct a graph G and a set of precedence constraints R such that there exists a feasible solution of the PCMCA problem in G if and only if Φ is satisfiable, i.e. if and only if the the underlying 3-SAT problem is satisfiable.

Let $G = (V, A)$ where $V = \{r\} \cup \{s\} \cup \{s'\} \cup \{t\} \cup C$, with C, A and R defined as follows.

$$C = \{v_{ik} : 1 \leq i \leq m, 1 \leq k \leq 3\}$$

$$A = \{(r, s), (r, s')\} \cup \{(s, v_{1j}), 1 \leq j \leq 3)\} \cup \{(v_{mj}, t), 1 \leq j \leq 3\}$$

$$\cup \{(v_{ij}, v_{i+1,k}), 1 \leq i < m, 1 \leq j, k \leq 3\} \cup \{(s', v_{ij}), 1 \leq i \leq m, 1 \leq j \leq 3\}$$

$$R = \{(t, s')\} \cup \{(v_{hk}, v_{ij}) : h > i, \; v_{hk} \text{ and } v_{ij} \text{ refer to the same variable, but exactly one of the two literals is negated}\}$$

Note that C contains $3m$ vertices, one for each literal of each clause C_i, with all

arcs having an equal positive cost. The three sets C, A, and R induce the graph shown in Figure 3.2. The set of precedence constraints, besides (t, s'), is between two vertices that refer to the same literal, but exactly one of the two literals is negated. If a feasible solution T of the PCMCA problem can be found in G, this implies that:

1. no path from s' to t exists in T
2. in any (rooted) path there is no pair of vertices corresponding to a variable and its negation
3. there is a unique path P from r to t which passes through s and through a vertex of each clause

The formula can be satisfied by assigning true values to all the literals corresponding to the vertices in $P \cap C$, and assigning false values to all the variables not associated with these literals. This satisfies all the clauses.

Conversely, if the formula is satisfied then each clause has at least one literal with true value, and no variable is assigned to both true and false (in different clauses). We construct a PCMCA feasible solution as follows. We start by building a path P from r to t which includes s and exactly one vertex from each clause, corresponding to a literal with true value. We complete the arborescence by adding (r, s') and (s', v) for each $v \notin P$. □

3.4 Mixed Integer Linear Programming Models

In this section we propose several MILP models for the PCMCA problem, using different techniques to model the precedence relationships between vertex pairs.

3.4.1 Multicommodity Flow Model

In this section we introduce a MILP model with a polynomial number of constraints for the PCMCA which modifies the *flow conservation constraints* of a *Multicommodity flow* model [11, 25].

The PCMCA problem can be modeled as a Multicommodity flow problem by modifying the flow conservation constraints, such that the flow originating from the root, that reaches every vertex k in the graph, cannot pass through a successor of k before reaching k (i is a successor of k if $(k,i) \in R$). The PCMCA problem can be modeled as a Multicommodity flow problem as follows.

Let x_{ij} be a variable associated with every arc $(i,j) \in A$ such that $x_{ij} = 1$ if $(i,j) \in T$, and 0 otherwise. Let y_{ij}^k be a variable associated with every vertex $k \in V \backslash \{r\}$ and every arc $(i,j) \in A$, such that $y_{ij}^k = 1$ if arc $(i,j) \in T$ on the path from r to k, and 0 otherwise. The PCMCA problem can be modeled as the following MILP model.

$$\text{minimize} \sum_{(i,j)\in A} c_{ij} x_{ij} \tag{3.1}$$

$$\text{subject to} \sum_{(i,j)\in A} x_{ij} = 1 \qquad \forall j \in V \backslash \{r\} \tag{3.2}$$

$$\sum_{\substack{(i,j)\in A: \\ (k,j)\notin R}} y_{ij}^k - \sum_{\substack{(j,i)\in A: \\ (k,j)\notin R}} y_{ji}^k = \begin{cases} 1 \text{ if } i = r \\ -1 \text{ if } i = k \\ 0 \text{ otherwise} \end{cases} \qquad \begin{array}{r} \forall k \in V \backslash \{r\}, \\ \forall i \in V : (k,i) \notin R \end{array} \tag{3.3}$$

$$y_{ij}^k \leq x_{ij} \qquad \forall k \in V \backslash \{r\}, (i,j) \in A \tag{3.4}$$

$$y_{ij}^k \in \{0,1\} \qquad \forall k \in V \backslash \{r\}, (i,j) \in A \tag{3.5}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A \quad (3.6)$$

Constraints (3.2) impose the first property of an arborescence namely that every vertex $v \in V \backslash \{r\}$ must have a single parent. Constraints (3.3) are the connectivity constraints (flow conservation constraints) which ensure that any feasible solution is an arborescence rooted at $r \in V$. The constraints also impose the precedence relationships, such that every path connecting vertex k to the root r does not include vertex j if $(k,j) \in R$. Constraints (3.4) enforce that arc $(i,j) \in A$ is part of the path from r to k if $x_{ij} > 0$. Finally, constraints (3.5) and (3.6) define the domain of the variables.

The set of constraints (3.3) can be added dynamically to the model when violated, which would reduce the number of constraints included in the model. This is beneficial since the set of constraints for a certain $k_1 \in V$ could satisfy the constraints for a certain $k_2 \in V$, thus making the model smaller and easier to solve in theory.

The separation procedure for the set of equalities (3.3) is described in Algorithm 2. Let $\bar{x} \in \{0,1\}^n$ correspond to the value of the x variables. A set of equalities (3.3) for some $k \in V \backslash \{r\}$ that is violated by the solution $\bar{x}$, can be detected by checking if vertex k is not reachable from the root, or by checking if a (t,k)-path exists in the solution $\bar{x}$ for $(k,t) \in R$. Such paths are checked in a directed graph $G' = (V, A')$, where $A' = \{(i,j) \in A \,|\, \bar{x}_{ij} = 1\}$.

Algorithm 2 Separation Procedure for Equalities (3.3)

```
1: procedure FIND_VIOLATED_FLOW_CONSERVATION_EQUALITY(G, x̄)
2:     Construct a directed graph G' = (V, A') such that:
3:     A' = {(i, j) ∈ A | x̄_ij = 1}
4:     for k ∈ V\{r} do
5:         Find a (r, k)-path P in G'
6:         if P = φ then
7:             return the set of violated equalities (3.3) for k
8:         end if
9:         for (k, t) ∈ R do
10:            Find a (t, k)-path P in G'
11:            if P ≠ φ then
12:                return the set of violated equalities (3.3) for k
13:            end if
14:        end for
15:    end for
16: end procedure
```

A (r, k)-path in G' can be found by traversing the path backwards starting from k, since there is only one unique path that can reach k from r in an integral solution. In this case, if vertex k is unreachable from r, then some vertex that has already been visited during the traversal will be visited again showing that the solution contains a cycle. If a cycle is detected (i.e. (r, k)-path $\notin A$) then the set of equalities (3.3) for k are added to the model. The same approach can be used to detect if a precedence violating (t, k)-path exists in G'. A (t, k)-path in G' can be found by traversing the path backwards starting from k, since there is only one unique path from r that can reach k in an integral solution. If vertex t is reachable by traversing the path backwards from k, and $(k, t) \in R$, then P contains a precedence violating (t, k)-path in G', otherwise P is an empty set. In this case the set of equalities (3.3) for k are added to the model.

3.4.2 Path-Based Models

In this section we describe two similar MILP models for the PCMCA problem that extend the *Set-Based* model introduced for the MCA problem (see section 2.2.1). The *Set-Based* model for the MCA problem is extended by adding polynomial sets of constraints that model the precedence relationships. In the two models, the precedence relationships are enforced by propagating a value through every path in the arborescence in order to check if the solution contains a precedence violating path. The two models are formally defined in the following two sections.

3.4.2.1 U^{st} Model

Let x_{ij} be a variable associated with every arc $(i,j) \in A$ such that $x_{ij} = 1$ if $(i,j) \in T$ and 0 otherwise. Let u_j^{st} be a variable associated with every vertex $j \in V$ and precedence relationship $(s,t) \in R$, such that $u_s^{st} = 0$ and $u_t^{st} = 1$ for all $(s,t) \in R$. A precedence relationship $(s,t) \in R$ can be satisfied by propagating the value of u_t^{st} along every path starting from t, and if a (t,s)-path exists in T, then we are propagating a value of one to vertex s and imposing that $u_s^{st} \geq 1$. However, we enforce $u_s^{st} = 0$, and therefore the solution is infeasible. Using the approach described previously the PCMCA problem can be modeled as the following MILP model.

$$\text{minimize} \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{3.7}$$

$$\text{subject to} \sum_{(i,j)\in A} x_{ij} = 1 \qquad \forall j \in V\backslash\{r\} \tag{3.8}$$

$$\sum_{(i,j)\in \delta^-(S)} x_{ij} \geq 1 \qquad \forall S \subseteq V\backslash\{r\} \tag{3.9}$$

$$u_s^{st} = 0 \qquad \forall (s,t) \in R \tag{3.10}$$

$$u_t^{st} = 1 \qquad \forall (s,t) \in R \tag{3.11}$$

$$u_j^{st} - u_i^{st} - x_{ij} \geq -1 \qquad \forall (s,t) \in R, (i,j) \in A \tag{3.12}$$

$$u_j^{st} \geq 0 \qquad \forall (s,t) \in R, j \in V \tag{3.13}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A \tag{3.14}$$

Constraints (3.8) enforce that every vertex other than the root must have exactly one incoming arc. Constraints (3.9) are the connectivity constraints that enforce every set of vertices $S \subseteq V \backslash \{r\}$ must be reachable from the root r. Constraints (3.10) and (3.11) set the values of u_s^{st} and u_t^{st} to 0 and 1 respectively, for all $(s,t) \in R$. Constraints (3.12) impose $u_j^{st} \geq u_i^{st}$ if $x_{ij} = 1$. Finally, constraints (3.13) and (3.14) define the domains of the variables.

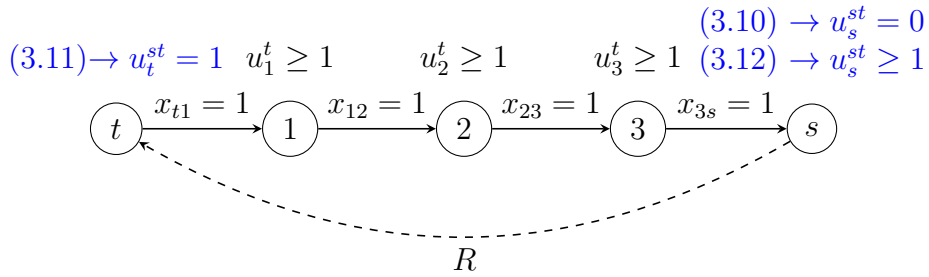

Figure 3.3: Value propagation demonstration over a violating (t,s)-path.

Figure 3.3 shows an example on how the value propagation occurs, where each black arc is weighted with its corresponding x_{ij} value, and the black dashed arrow is a precedence relationship $(s,t) \in R$. The u_j^{st} variable range is written above each vertex based on constraints (3.12). The figure shows an example of a precedence violating (t,s)-path, since constraints (3.12) impose that $u_s^{st} \geq 1$, while constraints (3.10) impose that $u_s^{st} = 0$ which means that the solution is infeasible.

3.4.2.2 U^t Model

Alternatively, the model described in the previous section can reformulated using a smaller number of variables and constraints following the same idea of propagating a value down every path starting from t in order to satisfy a precedence relationship $(s,t) \in R$. Let x_{ij} be a variable associated with every arc $(i,j) \in A$ such that $x_{ij} = 1$ if $(i,j) \in T$ and 0 otherwise. Let u_j^t be a variable associated with every vertex $j \in V$, and vertex $t \in V$ where t is part of a precedence relationship (i.e. $\exists (s,t) \in R$). Using the new set of variables u_j^t, the PCMCA problem can modeled as the following MILP model.

$$\text{minimize} \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{3.15}$$

$$\text{subject to} \sum_{(i,j)\in A} x_{ij} = 1 \qquad \forall j \in V\backslash\{r\} \tag{3.16}$$

$$\sum_{(i,j)\in \delta^-(S)} x_{ij} \geq 1 \qquad \forall S \subseteq V\backslash\{r\} \tag{3.17}$$

$$u_s^t = 0 \qquad \forall (s,t) \in R \tag{3.18}$$

$$u_t^t = 1 \qquad \forall t \in V : \exists (s,t) \in R \tag{3.19}$$

$$u_j^t - u_i^t - x_{ij} \geq -1 \qquad \forall t \in V : \exists (s,t) \in R, (i,j) \in A \tag{3.20}$$

$$u_j^t \geq 0 \qquad \forall t \in V : \exists (s,t) \in R, j \in V \tag{3.21}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A \tag{3.22}$$

Constraints (3.16) impose that every vertex other than the root must have a single parent. Constraints (3.17) are the connectivity constraints which enforce that for any set of vertices $S \subseteq V\backslash\{r\}$, there must be a path which connects r to S. Constraints (3.18) and (3.19) fix the values of u_s^t and u_t^t to 0 and 1 respectively, for

all $(s,t) \in R$, and $t \in V : \exists (s,t) \in R$. Constraints (3.20) impose $u_j^t \geq u_i^t$, if $x_{ij} = 1$. Finally, constraints (3.21) and (3.22) define the domains of the variables.

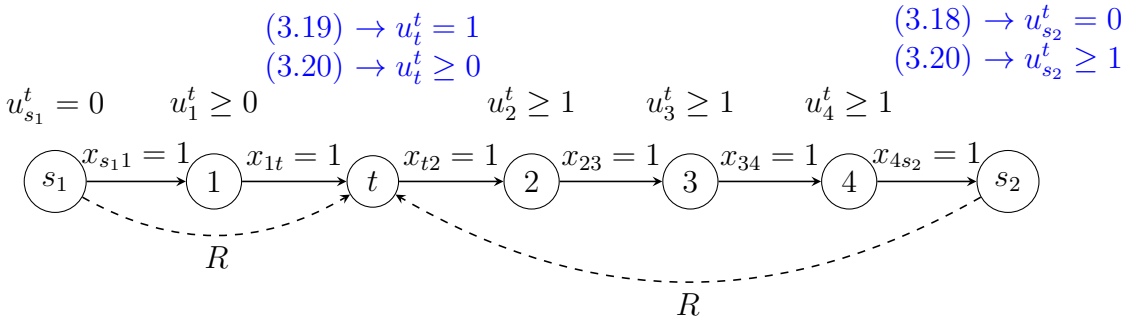

Figure 3.4: Value propagation demonstration using the reduced set of variables over a feasible path and a violating path.

Figure 3.4 shows an example on how the value propagation occurs using the reduced set of variables and constraints. In the figure, each black arc is weighted with its corresponding x_{ij} value, and black dashed arrow is a precedence relationship $(s,t) \in R$. The u_j^t variable range or value is written above each vertex based on constraints (3.20). The figure shows a feasible (s_1,t)-path, since constraints (3.20) impose that $u_t^t \geq 0$, and constraints (3.19) impose that $u_t^t = 1$. The figure also shows a precedence violating (t,s_2)-path, since constraints (3.20) impose that $u_s^{st} \geq 1$, while constraints (3.18) impose that $u_{s_2}^t = 0$ which means that the solution is infeasible.

Note that the set of inequalities (3.12) and (3.20) sometimes fail to detect a violating (t,s)-path in a fractional solution due to the diminishing of the propagated value along that path, but are always able to detect violating paths for integer solutions. An example of a fractional solution containing a violating path that cannot be detected by the set on inequalities (3.12) and (3.20) is shown in figure 3.5. The figure shows a fractional solution, where each arc is weighted by its corresponding x_{ij} variable value, and the black dashed arc represents the precedence relationship $(s,t) \in R$. The value of the variable u_j^{st} or u_j^t is shown next to each vertex. We can see in the figure that the violating path is not detected because a value of 0 is

propagated down to vertex s from t.

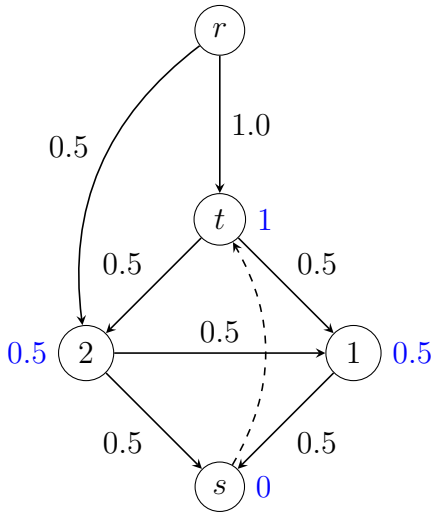

Figure 3.5: Example of a fractional solution that violates a precedence relationship $(s,t) \in R$, and how the violation cannot be detected by the Path-Based model.

The set of constraints (3.20) can be initially relaxed and iteratively added to the model when they are violated. The separation procedure for the set of constraints (3.20) is described in Algorithm 3. Let $\bar{x} \in \{0,1\}^n$ correspond to the value of the x variables. An inequality (3.20) that is violated by the solution $\bar{x}$ for some $(s,t) \in R$ can be detected by finding a (t,s)-path in a directed graph $G' = (V, A')$, where $A' = \{(i,j) \in A \,|\, \bar{x}_{ij} = 1\}$.

Algorithm 3 Separation Procedure for Inequalities (3.20)

procedure FIND_VIOLATED_u^t_VALUE_PROPAGATION_INEQUALITY(G, $\bar{x}$)
    Construct a directed graph $G' = (V, A')$ such that:
    $A' = \{(i,j) \in A \,|\, \bar{x}_{ij} = 1\}$
    for $(s,t) \in R$ **do**
        Find a (t,s)-path P in G'
        if $P \neq \phi$ **then**
            return the set of violated inequalities $u_j^t - u_i^t - x_{ij} \geq -1 \, \forall (i,j) \in P$
        end if
    end for
end procedure

A (t, s)-path in G' can be detected by traversing the path backwards starting from s, as there is only one unique path that can reach t in a solution that is an arborescence. If vertex t is reached starting from s and $(s, t) \in R$ then P contains a precedence violating (t, s)-path in G', otherwise P is an empty set. In this case the set of violated inequalities (3.20) are added to the model for all $(i, j) \in P$.

Note that the same separation procedure can be applied to detect a violated inequality (3.12) by replacing the inequality at line 7, by inequality the $u_j^{st} - u_i^{st} - x_{ij} \geq -1$.

3.4.3 Flow-Based Models

In this section we describe two similar MILP models for the PCMCA problem that extend the flow-based model for the MCA problem described in section 2.2.2. The two models described utilizes the two sets of precedence-enforcing constraints described in the previous section.

3.4.3.1 *Compact-U^{st}* Model

Let x_{ij} be a variable associated with every arc $(i, j) \in A$ such that $x_{ij} = 1$ if $(i, j) \in T$ and 0 otherwise. Let y_{ij} be a variable associated with every arc $(i, j) \in A$ that is equal to the amount of flow passing through that arc. Let u_j^{st} be a variable associated with every vertex $j \in V$ and precedence relationship $(s, t) \in R$, such that $u_s^{st} = 0$ and $u_t^{st} = 1$ for all $(s, t) \in R$. The PCMCA problem can be modeled as the following MILP model.

$$\text{minimize} \sum_{(i,j) \in A} c_{ij} x_{ij} \tag{3.23}$$

$$\text{subject to} \sum_{(i,j)\in A} y_{ij} - \sum_{(j,i)\in A} y_{ji} = \begin{cases} -1 & i \neq r \\ n-1 & i = r \end{cases} \qquad \forall i \in V \quad (3.24)$$

$$x_{ij} \geq \frac{y_{ij}}{n-1} \qquad \forall (i,j) \in A \quad (3.25)$$

$$u_s^{st} = 0 \qquad \forall (s,t) \in R \quad (3.26)$$

$$u_t^{st} = 1 \qquad \forall (s,t) \in R \quad (3.27)$$

$$u_j^{st} - u_i^{st} - x_{ij} \geq -1 \qquad \forall (s,t) \in R, (i,j) \in A \quad (3.28)$$

$$u_j^{st} \geq 0 \qquad \forall (s,t) \in R, j \in V \quad (3.29)$$

$$x_{ij} \in \{0,1\} \qquad \forall (i.j) \in A \quad (3.30)$$

$$y_{ij} \in \mathbb{Z}^+ \qquad \forall (i.j) \in A \quad (3.31)$$

Constraints (3.24) are the flow-connectivity constraints which enforce every vertex to be reachable from the root r, no arcs enter the root r, and that any feasible solution is an arborescence. Constraints (3.25) impose that the value of x_{ij} must be between 0 and 1, if the value of y_{ij} is nonzero. The value of y_{ij} is divided by $n-1$ since (3.24) $y_{ij} \leq n-1$, which would restrict the value of x_{ij} to be between 0 and 1. Constraints (3.26) and (3.27) set the values of u_s^{st} and u_t^{st} to 0 and 1 respectively, for all $(s,t) \in R$. Constraints (3.28) propagate the value of u_i^{st} down to u_j^{st} if $x_{ij} = 1$. Finally, constraints (3.29)-(3.31) define the domain of the variables.

3.4.3.2 *Compact-*U^t Model

Alternatively, the model described in the previous section can reformulated using a smaller number of variables and constraints following the idea described previously in section 3.4.2.2. Let u_j^t be a variable associated with every vertex $j \in V$, and vertex $t \in V$ where t is part of a precedence relationship (i.e. $\exists (s,t) \in R$). Using

the reduced set of variables u_j^t, the PCMCA problem can modeled as the following MILP model.

$$\text{minimize} \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{3.32}$$

$$\text{subject to} \sum_{(i,j)\in A} y_{ij} - \sum_{(j,i)\in A} y_{ji} = \begin{cases} -1 & i \neq r \\ n-1 & i = r \end{cases} \qquad \forall i \in V \tag{3.33}$$

$$x_{ij} \geq \frac{y_{ij}}{n-1} \qquad \forall (i,j) \in A \tag{3.34}$$

$$u_s^t = 0 \qquad \forall (s,t) \in R \tag{3.35}$$

$$u_t^t = 1 \qquad \forall t \in V : \exists (s,t) \in R \tag{3.36}$$

$$u_j^t - u_i^t - x_{ij} \geq -1 \qquad \forall t \in V : \exists (s,t) \in R, (i,j) \in A \tag{3.37}$$

$$u_j^t \geq 0 \qquad \forall t \in V : \exists (s,t) \in R, j \in V \tag{3.38}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i.j) \in A \tag{3.39}$$

$$y_{ij} \in \mathbb{Z}^+ \qquad \forall (i.j) \in A \tag{3.40}$$

Constraints (3.33) are the flow-connectivity constraints that enforce the following. For any vertex i, there must be a unique path which connects r to i. Any feasible solution must be a connected acyclic graph rooted at r (i.e. an r-arborescence). Constraints (3.34) restrict the value of x_{ij} to be between 0 and 1, if the value of y_{ij} is greater than zero. To impose that, the value of y_{ij} is divided by $n-1$ since (3.33) $y_{ij} \leq n-1$. Constraints (3.35) and (3.36) fix the values of u_s^t and u_t^t to 0 and 1 respectively, for all $(s,t) \in R$ and $t \in V : \exists (s,t) \in R$. Constraints (3.37) propagate the value of u_i^t down to u_j^t if $x_{ij} = 1$. Finally, constraints (3.38)-(3.40) define the domain of the variables.

3.4.4 Set-Based Model

In this section we describe a MILP model for the PCMCA problem that extend the set-based model for the MCA problem described in section 2.2.1.

The precedence relationships between pairs of vertices can be formulated by extending the connectivity constraints (2.4) for the MCA problem. When considering the set $S \subseteq V \backslash \{r\}$, we can extend the constraint for each $j \in S$, and enforce that at least one active arc must enter S, originating from the set of vertices that are allowed to precede j (i.e. $(j,i) \notin R$) on the path connecting r to j. The set of vertices V_j that are allowed to precede j on the same directed path connecting r to j is defined as:

$$V_j = \{i \in V \,|\, (j,i) \notin R\} \tag{3.41}$$

The PCMCA problem can be formally modeled as follows. Let x_{ij} be a variable associated with every arc $(i,j) \in A$ such that $x_{ij} = 1$ if $(i,j) \in T$ and 0 otherwise. The PCMCA problem can be modeled as the following MILP model, after extending the connectivity constraints (2.4) for the MCA problem.

$$\text{minimize} \sum_{(i,j)\in A} c_{ij} x_{ij} \tag{3.42}$$

$$\text{subject to} \sum_{(i,j)\in A} x_{ij} = 1 \qquad \forall j \in V \backslash \{r\} \tag{3.43}$$

$$\sum_{(i,k)\in \delta^-(S)} x_{ik} \geq 1 \qquad \forall j \in V \backslash \{r\}, \forall S \subseteq V_j \backslash \{r\} : j \in S \tag{3.44}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A \tag{3.45}$$

Constraints (3.43) implies the first property of an arborescence namely that every vertex $v \in V\backslash\{r\}$ must have a single parent. Constraints (3.44) model the connectivity constraints, that is every vertex $j \in V\backslash\{r\}$ must be reachable from the root. The same set of constraints also impose the precedence relationships. The inequalities imply that the resulting arborescence will not include a (t, s)-path in the resulting arborescence when $(s, t) \in R$. Finally, constraints (3.45) define the domain of the variables. Note that $V_j\backslash S$ contains at least r, while S contains at least j. Moreover, the set of constraints (3.44) reduce to the connectivity constraints (2.4) for the MCA problem when the set R of precedence relationships is empty. This is because when $R = \phi$, then $V_j = V$ for all $j \in V\backslash\{r\}$. The inequality (3.44) is the same inequality named *weak σ-inequality* considered by Ascheuer, Jünger and Reinelt [3] for the *Sequential Ordering problem* (SOP).

The separation procedure for the set of constraints (3.44) is described in Algorithm 4. Let $\bar{x} \in [0, 1]^n$ correspond to the value of the x variables. An inequality (3.44) that is violated by the solution $\bar{x}$ can be detected by computing a minimum (r, j)-cut C in a directed graph $D_j = (V_j, A')$, where $A' = \{(i, k) \in A \,|\, i, k \in V_j\}$. The cost c_{ik} of an arc $(i, k) \in A'$ is equal to $\bar{x}_{ik}$.

Algorithm 4 Separation Procedure for Inequalities (3.44)

procedure FIND_VIOLATED_PRECEDENCE_INEQUALITY(G, $\bar{x}$)
  for $j \in V\backslash\{r\}$ **do**
    Construct a directed graph $D_j = (V_j, A')$ such that:
    $V_j = \{i \in V : (j, i) \notin R\}$
    $A' = \{(i, k) \in A \,|\, i, k \in V_j\}$
    $c_{ik} = \bar{x}_{ik} \;\; \forall\, (i, k) \in A'$
    Calculate a minimum (r, j)-cut C in D_j
    if the cost of $C < 1$ **then**
      return the violated inequality $\sum_{(i,k)\in C} x_{ik} \geq 1$
    end if
  end for
end procedure

The value of a minimum (r, j)-cut C in D_j can reveal the following about the given solution $\bar{x}$:

1. If the cost of a minimum cut is equal to 0, then D_j does not contain a (r, j)-path. In this case, the solution does not contain a (r, j)-path, or contains a single or multiple (r, j)-paths, all of which pass through a successor of j.
2. If the cost of a minimum cut is in the range (0, 1), then D_j contains a (r, j)-path. In this case, the solution contains multiple (r, j)-paths, and at least one of them passes through a successor of j.
3. If the cost of a minimum cut is equal to 1, then D_j contains a single or multiple (r, j)-paths, although possibly some of them pass through a successor of j.

Note that if $\bar{x}$ contains fractional values, then in the first two cases, the minimum cut C defines an inequality (3.44) violated by $\bar{x}$. However, in the last case, a violated inequality (3.44) does not exist even if the solution $\bar{x}$ contains a precedence violating path. Therefore, although inequalities (3.44) are valid inequalities for the PCMCA problem, there are solutions that contain precedence violating paths, but satisfy inequalities (3.44).

A drawback of using the inequalities (3.44) in formulating the PCMCA problem, is the high computational complexity of the separation procedure, which has a complexity of $O(n^4)$, assuming it uses an $O(n^3)$ algorithm for computing a minimum (s, t)-cut in D_j [50].

Figure 3.6 shows an example on how the separation procedure works, where the figure on the left shows the solution graph, and the figure on the right shows the graph D_j that has a minimum (r, s)-cut of value one indicated by the black dashed curve. In both graphs, every arc cost is associated with the value of its respective x_{ij} variable. The example shows a candidate solution that contains a precedence

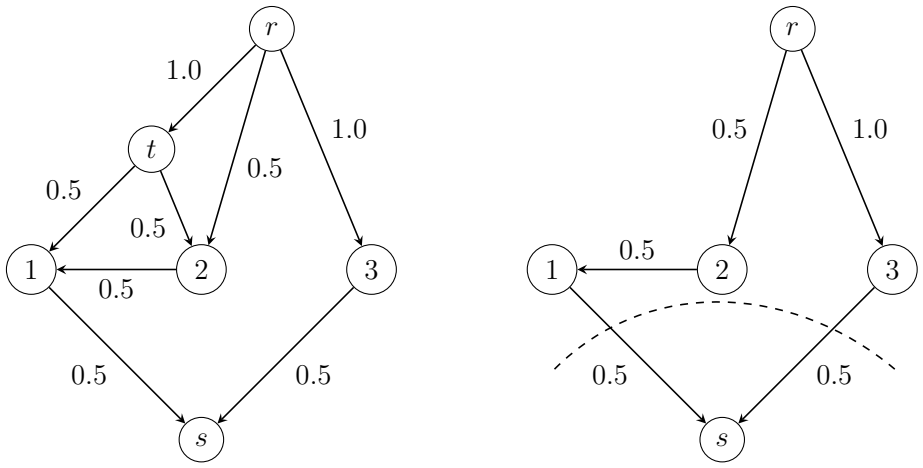

Figure 3.6: Example of separation procedure for inequalities (3.44).

violating path that violates the precedence relationship $(s, t) \in R$, but does not violate an inequality (3.44), since the value of the minimum cut is equal to one.

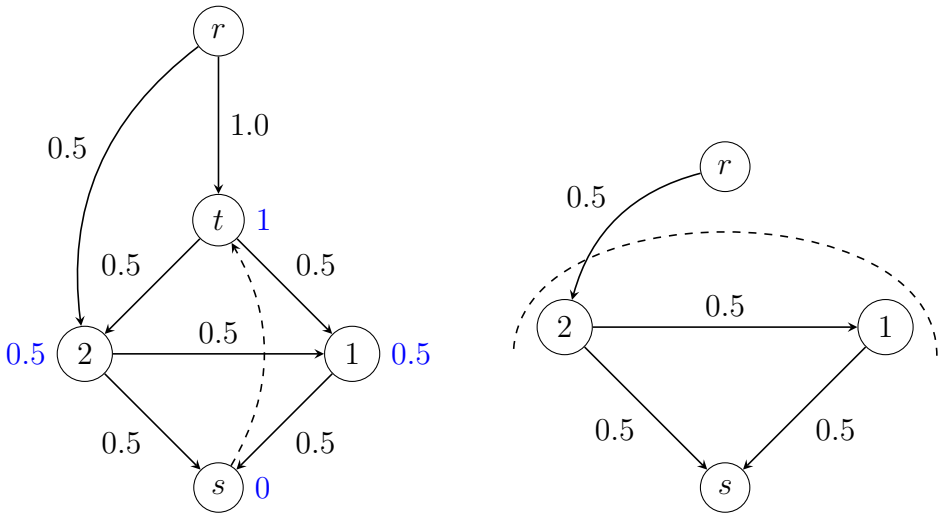

Figure 3.7: Example Comparing the Path-Based with the Set-Based model.

In section 3.4.2, we have shown an example of a fractional solution that contains a violating path that cannot be detected by the *Path-Based* model. However, the violating path can be detected for the same fractional solution by the *Set-Based* model after applying the separation procedure described in Algorithm 4 to the solution as shown in figure 3.7. The figure on the right shows how the *Set-Based* inequalities

are able to detect the violating path, since the graph has a minimum (r, s)-cut in D_j with value less than one.

3.5 A Branch-and-Bound Algorithm

In this section we present a *Branch-and-Bound* (B&B) algorithm for the PCMCA problem. The B&B algorithm is based on a *Lagrangian relaxation* [33] of the *Set-Based Model* discussed in section 3.4.4. The solving approach we propose is inspired by that discussed in Lucena [65] and Escudero et al. [28]. In particular, in [28] 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 (see chapter 3). Later, Toth and Vigo [84] 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.

The rest of this section is organized as follows. First we introduce a Lagrangian relaxation of the *Set-Based model*. Next, we describe the B&B algorithm we propose to solve the PCMCA problem.

3.5.1 A Lagrangian Relaxation

In this section we present a relaxation of the PCMCA obtained by relaxing the precedence-enforcing constraints (3.44), and leaving only the classical connectivity constraints of the MCA, and thus making the dual problem solvable in polynomial time.

Formally, the set of constraints (3.44) 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 (3.44) into the two sets of constraints results in the following model for the PCMCA problem.

$$\text{minimize} \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{3.46}$$

$$\text{subject to} \sum_{(i,j)\in A} x_{ij} = 1 \qquad \forall j \in V\backslash\{r\} \tag{3.47}$$

$$\sum_{(i,j)\in\delta^-(S)} x_{ij} \geq 1 \qquad \forall S \subseteq V\backslash\{r\} \tag{3.48}$$

$$\sum_{(i,j)\in\delta^-(S)} x_{ij} \geq 1 \quad \forall k \in V\backslash\{r\} : \exists (s,k) \in R, \forall S \subseteq V_k : k \in S \tag{3.49}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A \tag{3.50}$$

Constraints (3.48) are now the connectivity constraints that enforce every vertex to be reachable from the root. Constraints (3.49) 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 (3.49), thus reducing the problem to a MCA. The MCA can be solved in polynomial time using different algorithms (see section 2.3).

A stronger lower bound can however be obtained by adding constraints (3.49) to the objective function (3.46) in a Lagrangian relaxation fashion (see [33]).

3.5.1.1 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 the PCMCA can be obtained by introducing the precedence-enforcing constraints (3.49) in the objective function, with their corresponding Lagrangian multipliers λ. The following Lagrangian relaxation of the PCMCA can be obtained.

$$\text{minimize} \sum_{(i,j)\in A} c_{ij}x_{ij} + \sum_{\substack{k\in V\backslash\{r\}:\\ \exists(s,k)\in R}} \sum_{\substack{S\subseteq V_k\backslash\{r\}:\\ k\in S}} \lambda_S^k \left(\sum_{\substack{(i,j)\in A:\\ i\in V_k\backslash S, j\in S}} 1 - x_{ij} \right) \tag{3.51}$$

$$\text{subject to} \sum_{(i,j)\in A} x_{ij} = 1 \qquad \forall j \in V\backslash\{r\} \tag{3.52}$$

$$\sum_{(i,j)\in\delta^-(S)} x_{ij} \geq 1 \qquad \forall S \subseteq V\backslash\{r\} \tag{3.53}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A \tag{3.54}$$

Using a modified cost matrix c' defined as:

$$c'_{ij} = c_{ij} - \sum_{\substack{k\in V\backslash\{r\}:\\ i\in V_k}} \sum_{\substack{S\subseteq V_k\backslash\{r\}:\\ k,j\in S, i\notin S}} \lambda_S^k \tag{3.55}$$

the objective function (3.51) can be rewritten as the following.

$$\text{minimize} \sum_{(i,j)\in A} c'_{ij}x_{ij} + \sum_{k\in V\backslash\{r\}} \sum_{\substack{S\subseteq V_k\backslash\{r\}:\\ k\in S}} \lambda_S^k \tag{3.56}$$

For any given set of nonnegative multipliers λ, the optimal solution of the La-

grangian relaxation is a valid lower bound for the PCMCA [33]. 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 in section 2.3.

3.5.1.2 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 3.5.2), 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 5 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 3.6.2. 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.

Algorithm 5 Subgradient Method for Solving the Lagrangian Relaxation

1: **procedure** ComputeLB(λ, c, M, α)
2: $m = 0$
3: $LB = 0$
4: **do**
5: $x^\lambda = L(c, \lambda)$
6: $LB = \max(LB, c'x^\lambda + \lambda)$
7: $\bar{P} = \text{Separate}(x^\lambda)$
8: **for all** $p = \{t, 1, 2, \ldots, s\} \in \bar{P}$ **do**
9: $S = p \backslash \{t\}$
10: $\lambda_S^s = 0$
11: $\lambda = \lambda \cup \lambda_S^s$
12: **end for**
13: **for all** $\lambda_S^k \in \lambda$ **do**
14: $g = 1 - \sum_{\substack{(i,j)\in A: \\ i\in V_k \backslash S, j \in S}} x_{ij}^\lambda$
15: $\lambda_S^k = \max(0, \lambda_S^k + \alpha g)$
16: **end for**
17: $m = m + 1$
18: **while** $\bar{P} \neq \phi$ and $m < M$
19: **end procedure**

Step 5 finds an optimal solution of the MCA instance based on the objective function (3.56). Step 6 updates the value of the lower bound as the maximum between the current LB value, and the value of the objective function (3.56). 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 creates the Lagrangian multiplier λ_S^s and sets its initial value to zero, however it can be initialized to any non-negative initial value. Step 11 adds the corresponding Lagrangian multiplier λ_S^s to the set of multipliers λ. Steps 14 and 15 update the value of the Lagrangian multiplier λ_S^k. The value of step size $\alpha \in \mathbb{R}^+$ is a user-defined value, and is passed as a parameter.

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 5 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 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 3.5.4).

3.5.2 Branch-and-Bound Algorithm Data Structures

In this section we describe the data-structures used inside the B&B algorithm proposed for the exact solution of the PCMCA problem.

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.

3.5.2.1 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 (3.46).
- *SearchTreeNodes*: is a minimum priority queue that contains the set of all search-tree nodes that are currently unexplored in the search-tree, and is ordered by the lower bound value of the elements it contains.

3.5.2.2 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 (3.56) 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.

3.5.3 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 5, 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.

3.5.4 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 explores 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,

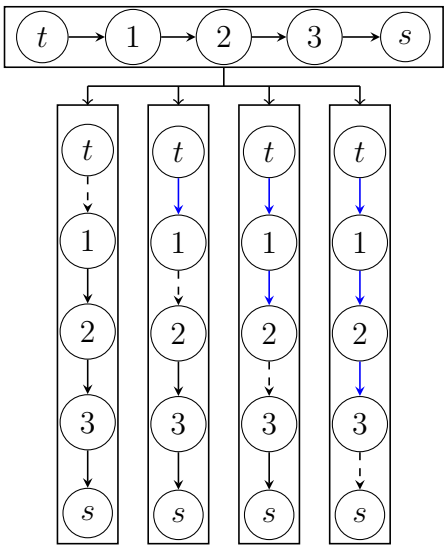

Figure 3.8: An example of a search-tree node being expanded into 4 new search-tree nodes.

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*.

Figure 3.8 shows an example of how a search-tree node is expanded. Each search-

tree node is represented by a rectangle, dashed arcs indicate forbidden arcs, and blue arcs indicate arcs that are imposed in the solution. In this example, we have the violating path $\{(t,1),(1,2),(2,3),(3,s)\}$. For each arc in the 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 selected violating path.

3.5.5 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 Figure 3.9. In the figure, dashed arrows indicate a precedence relationship, blue arrows indicate imposed arcs, and all green arrows indicate forbidden arcs. Based on the set of precedence relationships R, 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. Furthermore, an arc $(i,j) \in A$ is forbidden to appear in the solution, 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 (already explored) in the search tree reaches 2000 nodes, the MILP model introduced in Section 3.4.4 is solved for the original problem instance. The

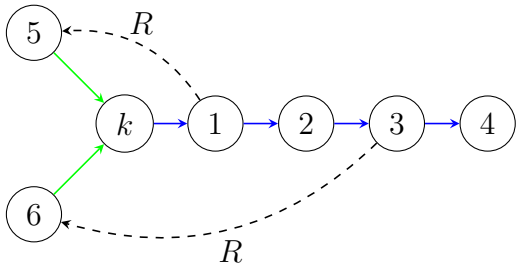

Figure 3.9: An example which shows an additional set of arcs that are forbidden to appear in the solution when a certain path rooted at k is imposed in the solution.

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.

3.6 Computational Results

In this section we present the experiments conducted to evaluate the several models proposed in section 3.4, and the B&B algorithm introduced in section 3.5. The computational experiments are based on the benchmark instances of TSPLIB [75], SOPLIB [69], and COMPILERS [79] originally proposed for the Sequential Ordering Problem [27]. The benchmark sets contain a total of 116 instances, ranging in size between 9 and 700 vertices, with an average of 248 vertices. A subset of the benchmark instances has been however modified to avoid a 0-cost solution. Such a solution exists because in the original instances there is a 0-cost arc from the root to every other node, and therefore we can connect every node directly to the root to achieve such a solution. The instances have been modified by randomly generating the weight of those arcs to be between 1 and the maximum weighted arc in the original graph.

All the experiments are performed on a laptop with an Intel i7-8550U processor

running at 1.8 GHz with 8 GB of RAM. The MILPs are solved using CPLEX 12.8 [53]. CPLEX is run with the two parameters *NodeSelect* and *MIP emphasis* set to *BestBound* (same as the branch selection strategy used by the B&B algorithm) and *MIPEmphasisOptimality* respectively, and single threaded standard Branch-and-Cut (B&C) algorithm is applied for solving the MILP models, as the B&B algorithm also uses a single thread. A time limit of 3 hours is set for the computation time for each computational method/instance. The models have been implemented in C++ 11, and are compiled with Microsoft C/C++ Optimizing Compiler v19.

For the rest of this chapter we will be referring to the *Mutlicommodity Flow Model* from section 3.4.1 as *MCF*. The *Flow-Based* model from section 3.4.3.1, that uses the set of u_j^{st} variables is referred to as *Compact-*U^{st}. The *Flow-Based* model from section 3.4.3.2, that uses the set of u_j^t variables is referred to as *Compact-*U^t. The *Path-Based* model from section 3.4.2.1, that uses the set of u_j^{st} variables is referred to as U^{st}. The *Path-Based* model from section 3.4.2.2, that uses the set of u_j^t variables, is referred to as U^t. Finally, the *Set-Based* model from section 3.4.4 is referred to as *Set-Based*.

3.6.1 The MILP Models

In this section we discuss the experiments conducted to evaluate the several models proposed in section 3.4. In section 3.6.1.1 we discuss the performance of the linear relaxation of the MILP models introduced in section 3.4, while in section 3.6.1.2 we discuss the performance of the integer program (IP) of those models.

3.6.1.1 The Linear Relaxation of the Models

An overview of the results shows that the solver optimally solves 98% of the instances, with a 0.002% average optimality gap using the model *MCF*. Such a results

is to be expected from a model that uses a Multicommodity flow formulation, however such models are generally computationally expensive to solve as will be shown later. Considering the rest of the models, the solver optimally solves 57% of the instances, with an average optimality gap of 1.2% using the model *Compact-U^{st}*. Using the model *Compact-U^{t}* the solver optimally solves 55% of the instances, with an average optimality gap of 3.1%. Using the model U^{st} the solver optimally solves 47% of the instances, with an average optimality gap of 2.1%. Using the model U^{t} the solver optimally solves 61% of the instances, with an average optimality gap of 0.9%. Finally, using the model *Set-Based* the solver optimally solves 68% of the instances, with an average optimality gap of 1.7%.

In terms of the number of cuts that are dynamically added to the model's linear relaxation, the solver adds 2846 cuts on average, with a median of 548, and a standard deviation of 6014, when solving the model *MCF*. Solving the model U^{st}, the solver adds 182 cuts on average (a 93.6% decrease), with a median of 46, and a standard deviation of 477. Solving the model U^{t}, the solver adds 81 cuts on average (a 97.2% decrease), with a median of 26, and a standard deviation of 162. Finally, solving the model *Set-Based*, the solver adds 71 cuts on average (a 97.5% decrease), with a median of 21, and a standard deviation of 228. For the two models *Compact-U^{st}* and *Compact-U^{t}*, the same set of precedence-enforcing constraints are not added dynamically in the two compact models, as preliminary experiments clearly showed an increase in the solution time.

In terms of computation time, solving the model *MCF* has an average solution time of 219 seconds, with a median of 3.1, and a standard deviation of 651.2 seconds. Solving the model *Compact-U^{st}* has an average solution time of 18 seconds (a 91.7% decrease), with a median of 4.4, and a standard deviation of 46.8 seconds. Solving the model *Compact-U^{t}* has an average solution time of 15 seconds (a 93.2% decrease), with a median of 4.5, and a standard deviation of 22.5 seconds. Solving the model

U^{st} has an average solution time of 55 seconds (a 74.9% decrease), with a median of 1.5, and a standard deviation of 309.6 seconds. Solving the model U^t has an average solution time of 13 seconds (a 94.1% decrease), with a median of 1.2, and a standard deviation of 78.2 seconds. Finally, solving the model *Set-Based* has an average solution time of 16 seconds (a 92.7% decrease), with a median 0.3, and a standard deviation of 93.1 seconds. The summary of the results is shown in table 3.1, where column *Optimal* reports the percentage of instances (for a total of 116 instances) that where solving the linear relaxation results in an optimal integer solution.

Table 3.1: Summary of the average results for the linear relaxation of the models *MCF, Compact-U^{st}, Compact-U^t, U^{st}, U^t* and *Set-Based*.

Model	Gap	Cuts	Time [s]	Optimal
MCF	0.002%	2846	219	98.3%
Compact-U^{st}	1.200%	-	18	57.8%
Compact-U^t	3.100%	-	15	55.2%
U^{st}	2.100%	182	55	47.4%
U^t	0.900%	81	13	61.2%
Set-Based	1.700%	71	16	68.1%

The results of optimality gap clearly shows that the model *MCF* has the smallest average optimality gap, and is able to optimally solve the majority of the instances at the root node of the search decision-tree. However, the average solution time of the model *MCF* and the number of cuts that are dynamically added to the model are much larger compared to the rest of the models. On the other hand, the model U^t achieves the second smallest average optimality gap, and the smallest average solution time. It is worth noting that the results of a model's linear relaxation generally does not fully show the performance of the IP of the same model, as will be shown in the next section. It is worth noting that the results of a model's linear

relaxation generally does not fully show the performance of the IP of the same model, as will be shown in the next section.

3.6.1.2 The IP Models

An overview of the results shows that the solver optimally solves all the instances within the time limit using the models *MCF*, U^{st}, U^{t}, and *Set-Based*. However, using the two models *Compact-*U^{st} and *Compact-*U^{t}, the solver fails to solve a single instance (*kro124p.3*) within the time limit of 3 hours.

In terms of the number of cuts that are dynamically added to the model, the solver adds 2849 cuts on average, with a median of 548, and a standard deviation of 6013, when solving the model *MCF*. Solving the model U^{st}, the solver adds 4751 cuts on average (a 66.8% increase), with a median of 106, and a standard deviation 21809. Solving the model U^{t}, the solver adds 222 cuts on average (a 92.2% decrease), with a median of 30, and a standard deviation of 548. Solving the model *Set-Based*, the solver adds 361 cuts on average (an 87.3% decrease), with a median of 26, and a standard deviation of 1622. For the two models *Compact-*U^{st} and *Compact-*U^{t} the same set of precedence-enforcing constraints are not added dynamically in the two compact models, as preliminary experiments clearly showed an increase in the solution time.

In terms of the number of nodes generated by the solver in the search decision-tree, solving the model *MCF* generates 0 nodes on average as the majority of the instances are solved at the root node of the search decision-tree. Solving the model *Compact-*U^{st}, the solver generates 1007 nodes on average, with a median of 0, and a standard deviation of 5663.6 nodes. Solving the model *Compact-*U^{t}, the solver generates 1480 nodes on average, with a median of 0, and a standard deviation of 8255.9 nodes. Solving the model U^{st}, the solver generates 5588 nodes on average, with a median of 2, and a standard deviation of 28372.2 nodes. Solving the model

U^t, the solver generates 2432 nodes on average, with a median of 0, and a standard deviation of 18465.2 nodes. Finally, Solving the model *Set-Based*, the solver generates 77 nodes on average, with a median of 0, and a standard deviation of 529.1 nodes.

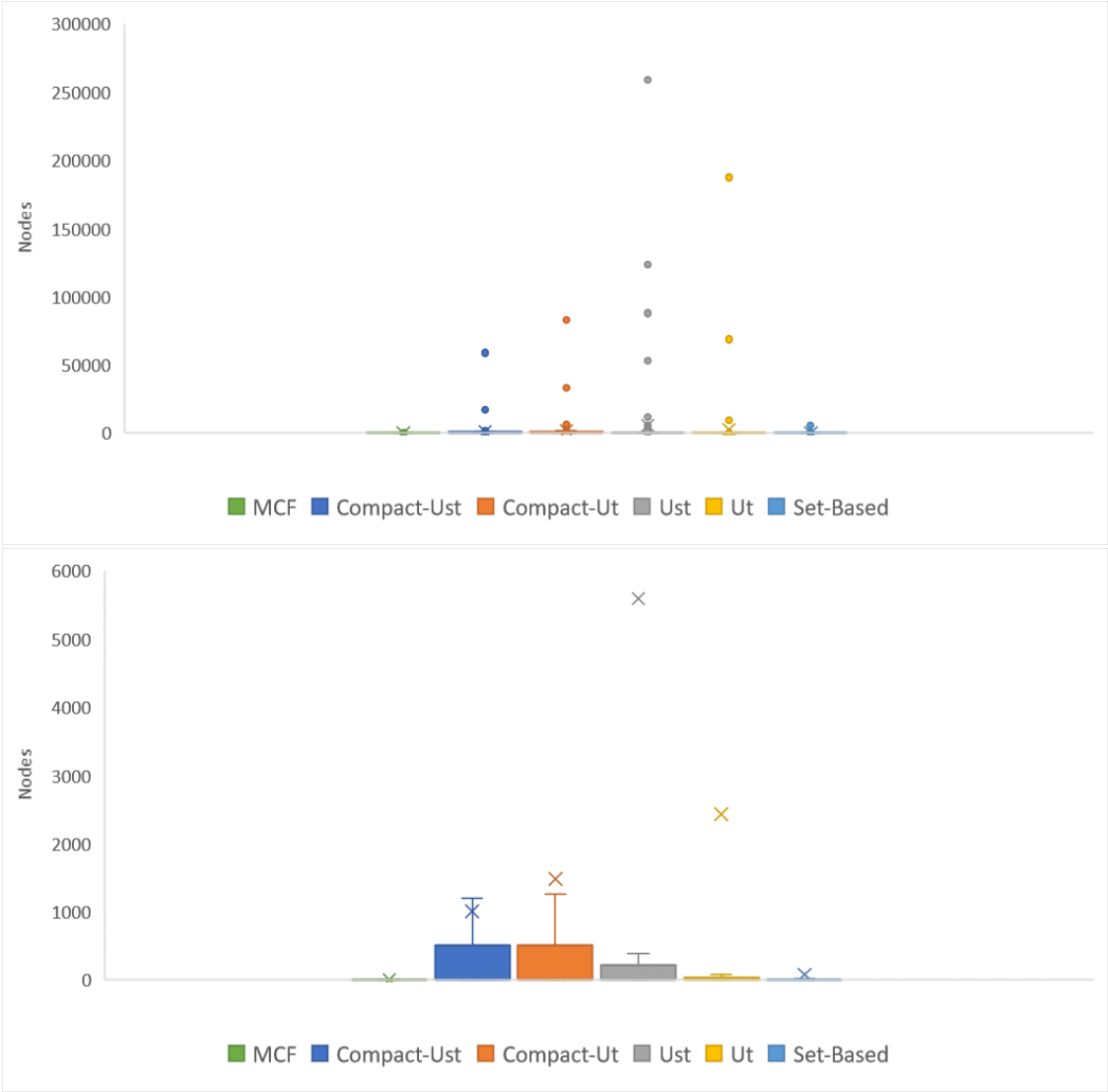

Figure 3.10: Distribution of the number of nodes generated for all the 116 instances.

Figure 3.10 shows the distribution of the number of nodes generated by the solver for each model, where the figure on the right excludes outlier points. We can see in the figure, that in general the *Set-Based* model generates the least amount of nodes

after the *MCF* model, even when we consider or exclude outliers points. However, the U^t model sometimes generates a smaller number of nodes, which depends on the structure of the solutions encountered after solving the linear relaxation, where each model is able to detect the violating path where another model fails.

In terms of solution time, solving the model *MCF* has an average solution time of 220 seconds, with a median of 3.3, and a standard deviation of 651.3 seconds. Solving the model *Compact-*U^{st} has an average solution time of 173 seconds (a 21.4% decrease), with a median of 19.9, and a standard deviation of 562.0 seconds. Solving the model *Compact-*U^t has an average solution time of 131 seconds (a 40.5% decrease), with a median of 23.7, and a standard deviation of 387.1 seconds. Solving the model U^{st} has an average solution time of 276 seconds (a 25.5% increase), with a median of 2.9, and a standard deviation of 1120.7 seconds. Solving the model U^t has an average solution time of 64 seconds (a 70.9% decrease), with a median of 1.5, and a standard deviation of 437.9 seconds. Finally, solving the model *Set-Based* has an average solution time of 27 seconds (a 87.7% decrease), with a median 0.8, and a standard deviation of 129.3 seconds.

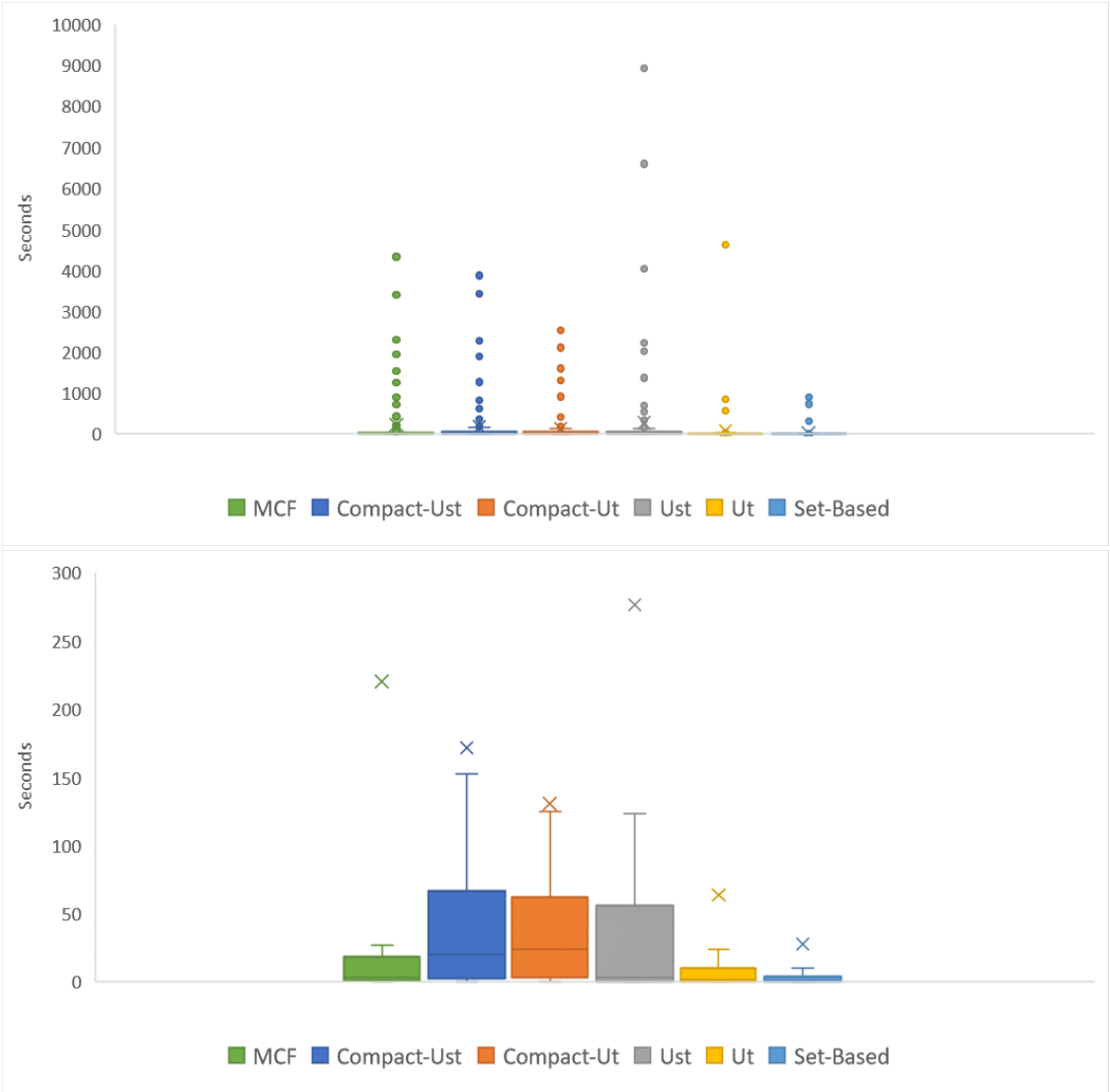

Figure 3.11: Distribution of solution times (in seconds) for all the 116 instances.

Figure 3.11 shows the distribution of solution times for each model, where the figure on the right excludes outlier points. We can see in the figure that the *Set-Based* model is in general the most efficient at solving the instances, also when considering the number of nodes generated in the search decision-tree, even when we consider or exclude outliers. However, the U^t model sometimes performs better on relatively large sized instances with either high or low density precedence relationships where the the separation procedure of the model becomes computationally

expensive (see instances *prob.100*, *R.200.100.15*, *jpeg.3195.85*). The summary of the average results is shown in table 3.2.

Table 3.2: Summary of the average results for the MILP models *MCF, Compact-U^{st}, Compact-U^t, U^{st}, U^t* and *Set-Based*.

Model	Nodes	Cuts	Time [s]
MCF	0	2849	220
Compact-U^{st}	1007	-	173
Compact-U^t	1480	-	131
U^{st}	5588	4751	276
U^t	2432	222	63
Set-Based	77	361	27

In conclusion, the *Set-Based* model in general outperforms the models considered. Indeed, it provides optimal solutions in substantially less time, and memory usage for the majority of the instances considered. The same cannot be said about the *Set-Based* model when linear relaxations only are considered, as the *Path-Based* (U^{st} and U^t models) can be solved much faster in most cases because of the fewer number of constraints, although it sometimes generates a looser estimate on the value of the optimal solution.

3.6.1.3 Overall Results

In this section we report the overall results of the models discussed throughout section 3.6.1. The overall results can be found in tables 3.3-3.8 where we report the following. Columns *Name* and *Size* report the name and size of the instance, $\rho(R)$ reports the density of arcs in the set of precedence relationships R, and is computed as $\frac{2 \cdot |R|}{|V|(|V|-1)}$. Column z^* reports the value of the optimal solution for that instance. For each model we report the following columns where applicable. Column *Cuts* reports the number of model-dependent cuts (inequalities) that are

dynamically added to the model, column *Nodes* reports the number of nodes in the search decision-tree, *Time [s]* reports the solution time in seconds. Column *Gap* reports the percentage relative difference between the optimal solution (z^*) of the PCMCA instance and the objective function value of the model's linear relaxation (z), computed as $\frac{z^*-z}{z^*}$. Finally, column *IP Gap* reports the optimality gap of the MILP model and is computed as $\frac{UB-LB}{UB}$, where LB is the best found lower bound on the optimal solution for that instance using the respective model, and UB is the best found solution for that instance using the respective model. The column is only reported for the model that does not optimally solve all instances within the time limit.

Table 3.3: PCMCA computational results for the linear relaxation of the models *MCF, Compact-U^{st}, Compact-U^{t}, U^{st}, U^{t}* and *Set-Based* for TSPLIB instances.

				Model														
Instance				MCF			Compact-U^{st}		Compact-U^{t}		U^{st}			U^{t}			Set-Based		
Name	Size	$\rho(R)$	z^*	Cuts	Time [s]	Gap	Time [s]	Gap	Time [s]	Gap	Cuts	Time [s]	Gap	Cuts	Time [s]	Gap	Cuts	Time [s]	Gap
br17.10	18	0.314	25	90	0.078	0.0000	0.270	0.140	0.160	0.412	25	0.032	0.000	17	0.047	0.000	21	0.015	0.000
br17.12	18	0.359	25	108	0.094	0.0000	0.230	0.061	0.080	0.454	25	0.047	0.000	19	0.060	0.000	22	0.016	0.000
ESC07	9	0.611	1531	9	0.031	0.0000	0.062	0.000	0.062	0.000	12	0.031	0.000	14	0.047	0.000	13	0.031	0.000
ESC11	13	0.359	1752	13	0.047	0.0000	0.078	0.000	0.110	0.027	5	0.031	0.000	3	0.032	0.000	1	0.031	0.000
ESC12	14	0.396	1138	14	0.031	0.0000	0.094	0.000	0.141	0.000	3	0.016	0.000	3	0.016	0.000	1	0.016	0.000
ESC25	27	0.177	1041	81	0.328	0.0000	0.078	0.000	0.156	0.000	14	0.062	0.000	22	0.078	0.000	31	0.063	0.000
ESC47	49	0.108	703	98	0.375	0.0000	0.812	0.000	1.062	0.000	102	0.484	0.003	105	0.253	0.000	142	0.547	0.000
ESC63	65	0.173	56	455	3.672	0.0000	2.328	0.000	0.609	0.000	14	0.329	0.000	270	2.484	0.000	42	0.218	0.000
ESC78	80	0.139	502	80	0.234	0.0000	1.594	0.000	3.406	0.000	12	0.094	0.000	3	0.050	0.000	1	0.047	0.000
ft53.1	54	0.082	3917	486	7.219	0.0000	0.630	0.024	0.700	0.023	123	1.172	0.004	72	0.760	0.003	78	0.328	0.002
ft53.2	54	0.094	3978	756	19.360	0.0000	0.580	0.021	0.630	0.021	3674	0.281	0.076	49	0.250	0.004	57	0.188	0.028
ft53.3	54	0.225	4242	648	9.766	0.0000	0.560	0.017	0.670	0.021	77	1.890	0.056	51	0.980	0.015	96	0.453	0.000
ft53.4	54	0.604	4882	378	1.750	0.0000	1.032	0.000	1.015	0.000	11	0.156	0.027	8	0.203	0.000	13	0.047	0.000
ft70.1	71	0.036	32846	497	11.422	0.0000	5.375	0.000	4.391	0.000	144	2.891	0.000	136	2.813	0.000	158	2.750	0.000
ft70.2	71	0.075	32930	568	10.641	0.0000	8.562	0.000	9.093	0.000	158	2.985	0.000	129	2.750	0.000	163	2.719	0.000
ft70.3	71	0.142	33431	1278	159.188	0.0000	1.500	0.005	1.580	0.004	45	0.750	0.024	131	3.550	0.003	66	0.265	0.020
ft70.4	71	0.589	35179	781	6.047	0.0000	1.060	0.000	13.906	0.000	217	13.015	0.006	21	0.110	0.026	30	0.094	0.021
rbg048a	50	0.444	204	50	0.110	0.0000	0.360	0.000	0.406	0.000	3	0.047	0.000	4	0.047	0.000	5	0.031	0.000
rbg050c	52	0.459	191	676	4.516	0.0000	1.344	0.000	1.609	0.000	35	0.313	0.000	16	0.141	0.000	11	0.047	0.000
rbg109	111	0.909	256	222	0.219	0.0000	0.797	0.000	1.125	0.000	47	11.578	0.000	6	0.109	0.000	14	0.094	0.000
rbg150a	152	0.927	373	304	0.359	0.0000	4.641	0.000	6.734	0.000	6	2.485	0.000	7	0.297	0.000	14	0.187	0.000
rbg174a	176	0.929	365	176	0.171	0.0000	2.750	0.019	3.970	0.019	56	29.610	0.003	32	1.047	0.000	22	0.297	0.000
rbg253a	255	0.948	375	765	4.828	0.0000	8.094	0.000	9.500	0.000	2	13.985	0.000	9	1.094	0.000	22	1.125	0.000
rbg323a	325	0.928	754	975	1.391	0.0000	29.750	0.000	23.890	0.000	16	1.547	0.000	5	1.391	0.000	26	1.047	0.000
rbg341a	343	0.937	22	3430	22.156	0.0000	17.390	0.010	12.890	0.010	395	23.344	0.033	30	15.530	0.011	89	3.031	0.000
rbg358a	360	0.886	595	1080	2.234	0.0000	40.515	0.000	40.750	0.000	4	0.312	0.000	26	21.343	0.000	67	5.812	0.000
rbg378a	380	0.894	559	3420	16.250	0.0000	15.750	0.007	13.860	0.009	464	16.079	0.039	29	31.250	0.000	21	1.829	0.045
kro124p.1	101	0.046	32597	1414	340.265	0.0000	11.760	0.026	12.080	0.026	39	0.734	0.060	222	0.520	0.001	95	1.782	0.000
kro124p.2	101	0.053	32851	1919	561.187	0.0000	10.300	0.021	12.110	0.023	23	0.578	0.069	228	3.030	0.006	109	1.828	0.006
kro124p.3	101	0.092	33779	2828	1531.000	0.0000	437.890	0.133	11.170	0.219	225	8.672	0.027	198	6.980	0.023	69	0.844	0.035
kro124p.4	101	0.496	37124	1818	50.015	0.0000	6.450	0.028	5.660	0.026	277	41.828	0.014	143	5.926	0.011	128	1.672	0.000
p43.1	44	0.101	2720	396	4.531	0.0000	1.060	0.048	1.060	0.048	112	0.594	0.127	384	5.125	0.000	68	0.187	0.104
p43.2	44	0.126	2720	528	9.297	0.0000	1.140	0.051	1.280	0.036	237	1.016	0.084	471	2.062	0.000	33	0.079	0.110
p43.3	44	0.191	2720	616	10.703	0.0000	1.640	0.033	0.970	0.577	111	0.547	0.144	270	1.531	0.000	77	0.188	0.075
p43.4	44	0.164	2820	440	2.157	0.0000	0.720	0.009	0.750	0.007	132	1.218	0.087	324	5.060	0.000	11	0.047	0.083
prob.100	100	0.048	649	827	276.521	0.0022	5.160	0.011	7.280	0.011	282	11.766	0.013	249	12.141	0.011	1840	622.437	0.002
prob.42	42	0.116	143	42	0.171	0.0000	0.407	0.000	0.328	0.000	29	0.125	0.000	32	0.125	0.000	2	0.032	0.000
ry48p.1	49	0.091	13092	203	2.093	0.0003	0.720	0.012	1.000	0.017	118	0.828	0.009	81	1.192	0.009	31	0.094	0.019
ry48p.2	49	0.103	13103	245	2.438	0.0000	1.260	0.011	6.560	0.024	128	1.031	0.006	90	1.110	0.005	58	0.235	0.000
ry48p.3	49	0.193	13886	784	26.594	0.0000	1.230	0.045	5.000	0.051	183	2.109	0.037	121	1.095	0.034	34	0.078	0.069
ry48p.4	49	0.588	15340	343	1.969	0.0000	0.810	0.046	3.300	0.054	65	2.531	0.072	93	0.981	0.034	65	0.172	0.058
Average				728	75.645	0.0001	15.287	0.019	5.392	0.052	187	4.808	0.025	101	3.259	0.005	94	15.878	0.017

Table 3.4: PCMCA computational results for the linear relaxation of the models *MCF*, *Compact-U^{st}*, *Compact-U^{t}*, U^{st}, U^{t} and *Set-Based* for SOPLIB instances.

Instance				Model														
				MCF			Compact-U^{st}		Compact-U^{t}		U^{st}			U^{t}			Set-Based	
Name	Size	$\rho(R)$	z^*	Cuts	Time [s]	Gap	Time [s]	Gap	Time [s]	Gap	Cuts	Time [s]	Gap	Cuts	Time [s]	Gap	Cuts	Time [s]	Gap
R.200.100.1	200	0.020	29	200	1.781	0.000	4.719	0.000	4.578	0.000	1	0.219	0.000	13	1.843	0.000	11	0.875	0.000
R.200.100.15	200	0.847	454	8600	713.657	0.000	17.690	0.019	4.050	0.020	1274	3235.391	0.057	168	13.365	0.048	85	1.079	0.139
R.200.100.30	200	0.957	529	2600	8.375	0.000	1.840	0.017	1.630	0.022	27	12.922	0.112	43	3.051	0.029	39	0.266	0.093
R.200.100.60	200	0.991	6018	0	0.031	0.000	1.141	0.000	1.016	0.000	0	3.593	0.000	0	0.157	0.000	0	0.094	0.000
R.200.1000.1	200	0.020	887	200	3.625	0.000	9.562	0.000	9.875	0.000	0	0.203	0.000	2	0.625	0.000	3	0.656	0.000
R.200.1000.15	200	0.876	5891	6000	102.359	0.000	3.280	0.010	2.640	0.010	170	203.234	0.043	30	3.750	0.049	35	0.766	0.056
R.200.1000.30	200	0.958	7653	1000	3.172	0.000	1.520	0.001	1.830	0.002	45	56.000	0.000	7	0.953	0.000	9	0.234	0.000
R.200.1000.60	200	0.989	6666	0	0.047	0.000	1.469	0.000	1.579	0.000	0	3.797	0.000	0	0.157	0.000	0	0.094	0.000
R.300.100.1	300	0.013	13	1200	84.782	0.000	10.515	0.000	3.360	0.000	0	0.500	0.000	22	5.313	0.000	14	2.25	0.000
R.300.100.15	300	0.905	575	16800	893.625	0.000	9.330	0.026	9.990	0.026	149	3.985	0.103	228	46.330	0.036	20	1.171	0.077
R.300.100.30	300	0.970	756	1800	8.125	0.000	3.630	0.007	4.470	0.005	57	1.672	0.000	8	1.313	0.000	27	0.562	0.000
R.300.100.60	300	0.994	708	300	0.390	0.000	19.672	0.000	18.656	0.000	57	1.531	0.000	11	1.718	0.000	2	0.297	0.000
R.300.1000.1	300	0.013	715	900	297.563	0.000	58.562	0.000	62.375	0.000	69	10.546	0.000	69	10.343	0.000	8	2.094	0.000
R.300.1000.15	300	0.905	6660	10800	421.860	0.000	6.020	0.006	7.410	0.006	65	0.812	0.060	75	15.082	0.009	136	2.61	0.008
R.300.1000.30	300	0.965	8693	1800	4.110	0.000	7.328	0.000	10.718	0.000	11	1.531	0.000	1	1.016	0.000	6	0.391	0.000
R.300.1000.60	300	0.994	7678	600	0.672	0.000	10.672	0.000	11.109	0.000	4	23.234	0.000	0	0.469	0.000	2	0.297	0.000
R.400.100.1	400	0.010	6	0	0.688	0.000	18.093	0.000	19.515	0.000	1	0.391	0.000	7	1.142	0.000	42	5.781	0.000
R.400.100.15	400	0.927	699	22400	2291.765	0.000	24.470	0.011	10.450	0.014	22	0.328	0.108	62	26.167	0.033	24	0.906	0.100
R.400.100.30	400	0.978	712	4800	8.172	0.000	21.110	0.000	24.875	0.000	58	10.156	0.000	2	8.078	0.000	14	1.656	0.000
R.400.100.60	400	0.996	557	0	0.234	0.000	10.265	0.000	11.031	0.000	2	0.219	0.000	1	0.181	0.000	0	0.328	0.000
R.400.1000.1	400	0.010	780	2400	1256.485	0.000	12.953	0.000	13.140	0.000	13	6.734	0.000	17	11.672	0.000	4	2.797	0.000
R.400.1000.15	400	0.930	7382	16000	569.484	0.000	8.380	0.019	7.260	0.019	27	0.625	0.085	42	31.662	0.023	78	5.375	0.022
R.400.1000.30	400	0.977	9368	3600	8.828	0.000	8.190	0.021	11.750	0.023	541	34.531	0.011	36	13.551	0.025	20	1.14	0.044
R.400.1000.60	400	0.995	7167	400	0.563	0.000	33.078	0.000	36.500	0.026	44	2.016	0.000	3	1.453	0.000	1	0.5	0.000
R.500.100.1	500	0.008	3	3000	1687.547	0.000	37.921	0.000	34.469	0.000	579	217.172	0.000	172	35.726	0.000	29	11.812	0.000
R.500.100.15	500	0.945	860	27000	1934.063	0.000	31.550	0.017	17.050	0.016	20	1.016	0.085	51	35.192	0.041	100	7.406	0.039
R.500.100.30	500	0.980	710	8000	197.953	0.000	68.609	0.000	61.734	0.000	333	14.453	0.031	16	23.592	0.006	19	0.797	0.066
R.500.100.60	500	0.996	566	500	1.078	0.000	43.265	0.000	43.234	0.000	0	0.687	0.000	0	0.625	0.000	1	0.844	0.000
R.500.1000.1	500	0.008	297	0	1.203	0.000	17.469	0.000	17.250	0.000	0	0.609	0.000	0	0.611	0.000	0	4.469	0.000
R.500.1000.15	500	0.940	8063	24000	1270.750	0.000	14.520	0.001	88.218	0.000	648	82.015	0.000	37	48.410	0.006	119	15.063	0.000
R.500.1000.30	500	0.981	9409	3500	4.922	0.000	30.516	0.000	32.954	0.000	28	11.141	0.000	2	7.985	0.000	11	3.125	0.000
R.500.1000.60	500	0.996	6163	500	1.360	0.000	36.984	0.000	40.718	0.000	0	0.671	0.000	0	0.634	0.000	1	0.875	0.000
R.600.100.1	600	0.007	1	3600	3397.079	0.000	81.797	0.000	81.532	0.000	858	659.156	0.000	1262	840.446	0.000	1455	733.375	0.000
R.600.100.15	600	0.950	568	19200	534.797	0.000	58.094	0.000	63.406	0.000	387	31.516	0.000	10	12.750	0.000	23	5.312	0.000
R.600.100.30	600	0.985	776	4200	46.375	0.000	38.810	0.007	43.580	0.007	263	13.484	0.017	0	15.578	0.000	24	2.375	0.000
R.600.100.60	600	0.997	538	0	0.610	0.000	24.453	0.000	24.672	0.000	0	0.359	0.000	0	0.265	0.000	0	0.906	0.000
R.600.1000.1	600	0.007	322	0	1.969	0.000	32.016	0.000	33.969	0.000	0	0.844	0.000	0	0.735	0.000	0	8.625	0.000
R.600.1000.15	600	0.945	9763	30600	1977.250	0.000	42.063	0.000	66.703	0.000	216	17.984	0.022	20	55.640	0.000	69	12.766	0.000
R.600.1000.30	600	0.984	9497	4200	24.015	0.000	48.735	0.000	97.906	0.000	14	7.219	0.000	3	3.714	0.000	13	2.969	0.000
R.600.1000.60	600	0.997	6915	0	0.625	0.000	33.422	0.000	34.578	0.000	1	0.406	0.000	1	0.125	0.000	0	0.922	0.000
R.700.100.1	700	0.006	2	4900	4327.026	0.000	117.281	0.000	35.234	0.000	0	1.250	0.000	0	1.152	0.000	616	314.875	0.000
R.700.100.15	700	0.957	675	12600	206.344	0.000	176.047	0.000	86.688	0.000	106	41.000	0.000	7	12.766	0.000	23	6.875	0.000
R.700.100.30	700	0.987	590	700	1.656	0.000	61.203	0.000	74.266	0.000	0	3.984	0.000	0	2.813	0.000	1	1.25	0.000
R.700.100.60	700	0.997	383	0	0.703	0.000	46.437	0.000	45.469	0.000	0	0.500	0.000	0	0.435	0.000	0	1.422	0.000
R.700.1000.1	700	0.006	611	0	3.031	0.000	57.156	0.000	61.797	0.000	3	1.625	0.000	7	5.156	0.000	0	13.891	0.000
R.700.1000.15	700	0.956	2792	2100	7.094	0.000	28.609	0.000	35.828	0.000	3	1.500	0.000	1	1.156	0.000	4	1.875	0.000
R.700.1000.30	700	0.986	2658	0	0.516	0.000	20.078	0.000	23.500	0.000	0	0.360	0.000	0	0.259	0.000	0	0.828	0.000
R.700.1000.60	700	0.997	1913	0	0.640	0.000	55.750	0.000	60.719	0.000	0	0.515	0.000	0	0.315	0.000	0	1.375	0.000
Average				5229	464.771	0.000	31.381	0.003	31.152	0.004	127	98.409	0.015	51	27.197	0.006	64	24.7135625	0.013

Table 3.5: PCMCA computational results for the linear relaxation of the models *MCF, Compact-U^{st}, Compact-U^t, U^{st}, U^t* and *Set-Based* for COMPILERS instances.

Instance				Model														
				MCF			Compact-U^{st}		Compact-U^t		U^{st}			U^t			Set-Based		
Name	Size	$\rho(R)$	z^*	Cuts	Time [s]	Gap	Time [s]	Gap	Time [s]	Gap	Cuts	Time [s]	Gap	Cuts	Time [s]	Gap	Cuts	Time [s]	Gap
gsm.153.124	126	0.970	185	378	0.375	0.000	1.610	0.000	1.125	0.029	180	0.578	0.000	26	0.297	0.000	49	0.125	0.011
gsm.444.350	353	0.990	1542	0	0.078	0.000	0.594	0.000	0.454	0.000	2	0.078	0.000	12	0.375	0.000	0	0.094	0.000
gsm.462.77	79	0.840	292	474	0.875	0.000	2.109	0.000	2.234	0.003	48	3.422	0.000	31	0.296	0.000	14	0.031	0.000
jpeg.1483.25	27	0.484	71	54	0.172	0.000	1.156	0.000	0.190	0.017	50	0.234	0.000	48	0.082	0.000	21	0.031	0.000
jpeg.3184.107	109	0.887	411	981	1.953	0.000	1.391	0.000	0.500	0.007	96	14.640	0.006	56	0.660	0.004	32	0.093	0.000
jpeg.3195.85	87	0.740	13	1392	6.328	0.000	3.390	0.088	3.610	0.000	2548	278.844	0.385	661	12.312	0.385	45	0.125	0.385
jpeg.3198.93	95	0.752	140	27343	15.531	0.000	17.390	0.000	28.203	0.000	2093	252.734	0.029	647	13.280	0.021	29	0.141	0.036
jpeg.3203.135	137	0.897	507	1918	4.109	0.000	1.090	0.016	0.980	0.017	104	47.578	0.004	88	1.751	0.004	18	0.094	0.022
jpeg.3740.15	17	0.257	33	34	0.125	0.000	0.220	0.045	0.240	0.046	72	1.782	0.030	24	0.067	0.030	17	0.031	0.000
jpeg.4154.36	38	0.633	74	228	0.734	0.000	0.310	0.041	0.220	0.041	52	0.641	0.050	42	0.160	0.051	43	0.063	0.000
jpeg.4753.54	56	0.769	146	224	0.469	0.000	4.391	0.000	2.641	0.000	154	2.766	0.007	67	0.342	0.010	38	0.062	0.007
susan.248.197	199	0.939	688	2587	8.110	0.000	0.750	0.008	0.610	0.008	75	76.329	0.003	54	2.451	0.002	21	0.125	0.000
susan.260.158	160	0.916	472	2400	4.860	0.000	1.340	0.010	0.780	0.010	20	12.156	0.017	75	2.080	0.006	33	0.141	0.000
susan.343.182	184	0.936	468	2760	6.829	0.000	1.520	0.007	1.470	0.006	201	194.188	0.010	123	6.842	0.009	47	0.203	0.010
typeset.10192.123	125	0.744	241	1500	10.359	0.000	2.740	0.037	1.910	0.041	14	4.859	0.103	103	2.890	0.016	93	0.5	0.000
typeset.10835.26	28	0.349	60	112	0.407	0.000	0.079	0.000	0.078	0.000	8	0.063	0.000	9	0.047	0.000	14	0.031	0.000
typeset.12395.43	45	0.518	125	270	1.000	0.000	1.250	0.000	0.280	0.013	37	0.531	0.005	28	0.125	0.000	27	0.078	0.000
typeset.15087.23	25	0.557	89	50	0.250	0.000	0.170	0.011	0.190	0.011	84	0.297	0.011	25	0.030	0.011	24	0.047	0.000
typeset.15577.36	38	0.555	93	76	0.204	0.000	0.516	0.000	0.468	0.000	7	0.031	0.000	6	0.062	0.000	4	0.015	0.000
typeset.16000.68	70	0.658	67	490	1.219	0.000	4.530	0.093	13.560	0.093	787	21.891	0.000	425	17.725	0.000	643	3.281	0.090
typeset.1723.25	27	0.245	54	216	1.375	0.000	0.340	0.092	0.440	0.989	88	0.203	0.056	52	0.160	0.056	19	0.031	0.056
typeset.19972.246	248	0.993	979	0	0.046	0.000	0.234	0.000	0.250	0.000	14	0.110	0.000	7	0.069	0.000	0	0.062	0.000
typeset.4391.240	242	0.981	837	2178	2.735	0.000	5.156	0.000	6.359	0.000	131	378.172	0.001	95	3.782	0.000	18	0.094	0.000
typeset.4597.45	47	0.493	133	94	0.235	0.000	0.234	0.000	0.297	0.000	16	0.437	0.000	16	0.079	0.000	7	0.031	0.000
typeset.4724.433	435	0.995	1819	3045	3.578	0.000	2.030	0.003	1.980	0.003	374	4.000	0.000	68	1.823	0.000	8	0.172	0.000
typeset.5797.33	35	0.748	93	35	0.094	0.000	0.407	0.000	0.297	0.000	85	0.234	0.000	37	0.125	0.000	9	0.032	0.000
typeset.5881.246	248	0.986	979	496	0.484	0.000	0.530	0.003	0.810	0.003	49	191.813	0.003	55	1.885	0.003	52	0.343	0.000
Average				1827	2.686	0.000	2.055	0.017	2.599	0.050	274	55.134	0.027	107	2.585	0.023	49	0.225	0.023

Table 3.6: PCMCA computational results for the MILP models *MCF*, *Compact-U^{st}*, *Compact-U^{t}*, U^{st}, U^{t} and *Set-Based* for TSPLIB instances.

Instance				Model																
				MCF			Compact-U^{st}			Compact-U^{t}			U^{st}			U^{t}			Set-Based		
Name	Size	$\rho(R)$	z^*	Nodes	Cuts	Time [s]	Nodes	Time [s]	IP Gap	Nodes	Time [s]	IP Gap	Nodes	Cuts	Time [s]	Nodes	Cuts	Time [s]	Nodes	Cuts	Time [s]
br17.10	18	0.314	25	0	90	0.078	1024	0.922	-	483	0.938	-	3	26	0.060	0	17	0.047	0	21	0.015
br17.12	18	0.359	25	0	108	0.094	44	1.125	-	516	1.563	-	3	26	0.063	15	20	0.094	0	22	0.016
ESC07	9	0.611	1531	0	9	0.031	0	0.062	-	0	0.047	-	0	12	0.031	0	14	0.047	0	13	0.031
ESC11	13	0.359	1752	0	13	0.047	0	0.078	-	6	0.078	-	0	5	0.031	0	3	0.032	0	1	0.031
ESC12	14	0.396	1138	0	14	0.031	0	0.094	-	0	0.079	-	0	3	0.016	0	3	0.016	0	1	0.016
ESC25	27	0.177	1041	0	81	0.328	0	0.078	-	0	0.093	-	0	14	0.062	0	22	0.078	0	31	0.063
ESC47	49	0.108	703	0	98	0.375	0	0.812	-	0	0.890	-	5	106	0.469	0	105	0.253	0	142	0.547
ESC63	65	0.173	56	0	455	3.672	0	2.328	-	0	0.985	-	0	14	0.329	0	270	2.484	0	42	0.218
ESC78	80	0.139	502	0	80	0.234	0	1.594	-	0	1.703	-	0	12	0.094	0	3	0.050	0	1	0.047
ft53.1	54	0.082	3917	0	486	7.219	966	10.282	-	1477	10.547	-	7	129	1.172	7	82	0.812	5	84	0.375
ft53.2	54	0.094	3978	0	756	19.360	481	17.094	-	1580	19.594	-	104	302	0.688	16	55	0.297	55	211	0.547
ft53.3	54	0.225	4242	0	648	9.766	616	10.469	-	361	10.093	-	122	416	2.547	33	82	1.156	0	96	0.453
ft53.4	54	0.604	4882	0	378	1.750	0	1.032	-	0	1.015	-	9	46	0.250	0	8	0.203	0	13	0.047
ft70.1	71	0.036	32846	0	497	11.422	0	5.375	-	0	4.391	-	1	144	2.828	0	136	2.813	0	158	2.750
ft70.2	71	0.075	32930	0	568	10.641	0	8.562	-	0	9.093	-	2	160	3.016	2	138	2.781	0	163	2.719
ft70.3	71	0.142	33431	0	1278	159.188	547	113.250	-	462	59.719	-	954	3061	63.171	280	288	6.531	145	2077	38.250
ft70.4	71	0.589	35179	0	781	6.047	6	7.860	-	0	13.906	-	53	457	13.438	37	217	1.515	369	1070	6.281
rbg048a	50	0.444	204	0	50	0.110	0	0.360	-	0	0.406	-	0	3	0.047	0	4	0.047	0	5	0.031
rbg050c	52	0.459	191	0	676	4.516	0	1.344	-	0	1.609	-	0	35	0.313	0	16	0.141	0	11	0.047
rbg109	111	0.909	256	0	222	0.219	0	0.797	-	0	1.125	-	0	47	11.578	0	6	0.109	0	14	0.094
rbg150a	152	0.927	373	0	304	0.359	0	4.641	-	0	6.734	-	0	6	2.485	0	7	0.297	1	14	0.219
rbg174a	176	0.929	365	0	176	0.171	438	20.312	-	16	25.391	-	2	57	29.609	0	32	1.047	1	22	0.313
rbg253a	255	0.948	375	0	765	4.828	0	8.094	-	0	9.500	-	0	2	13.985	0	9	1.094	0	22	1.125
rbg323a	325	0.928	754	0	975	1.391	0	29.750	-	0	23.890	-	0	16	1.547	0	5	1.391	0	26	1.047
rbg341a	343	0.937	22	0	3430	22.156	54	373.516	-	385	499.281	-	376	11958	278.859	60	40	23.547	0	89	3.031
rbg358a	360	0.886	595	0	1080	2.234	0	40.515	-	0	40.750	-	0	4	0.312	0	26	21.343	0	67	5.812
rbg378a	380	0.894	559	0	3420	16.250	523	615.093	-	510	181.703	-	543	4390	178.515	0	29	31.250	36	282	19.047
kro124p.1	101	0.046	32597	0	1414	340.265	500	44.594	-	504	43.313	-	47	312	1.844	6	234	0.703	0	95	1.782
kro124p.2	101	0.053	32851	0	1919	561.187	1459	810.547	-	1263	909.469	-	1433	801	11.203	1052	416	9.859	27	238	3.281
kro124p.3	101	0.092	33779	0	2828	1531.000	821	-	0.105	2966	-	0.094	258648	4253	6599.140	187246	1552	4625.841	98	656	7.469
kro124p.4	101	0.496	37124	0	1818	50.015	521	212.687	-	994	207.360	-	198	981	59.359	206	226	8.157	0	128	1.672
p43.1	44	0.101	2720	0	396	4.531	487	4.297	-	487	4.250	-	238	1202	4.203	0	384	5.125	128	692	1.765
p43.2	44	0.126	2720	0	528	9.297	491	9.859	-	496	15.281	-	119	589	1.781	0	471	2.062	237	1164	4.359
p43.3	44	0.191	2720	0	616	10.703	508	151.453	-	610	112.109	-	283	1113	2.829	0	270	1.531	134	598	1.437
p43.4	44	0.164	2820	0	440	2.157	1004	26.844	-	2016	25.437	-	198	926	3.516	99	435	5.516	353	1065	2.797
prob.100	100	0.048	649	11	1000	334.328	2265	389.265	-	2826	60.641	-	1428	2555	36.594	3633	2401	119.406	4	1962	743.969
prob.42	42	0.116	143	0	42	0.171	0	0.407	-	0	0.328	-	0	29	0.125	0	32	0.125	0	2	0.032
ry48p.1	49	0.091	13092	4	294	4.531	1338	85.531	-	1523	40.375	-	879	380	1.656	880	183	2.594	54	177	0.609
ry48p.2	49	0.103	13103	0	245	2.438	578	31.500	-	617	60.641	-	220	450	1.593	37	130	1.296	0	58	0.235
ry48p.3	49	0.193	13886	0	784	26.594	58651	3421.736	-	33288	2111.234	-	123233	2793	638.344	68658	1006	567.140	146	634	2.156
ry48p.4	49	0.588	15340	0	343	1.969	1269	357.203	-	718	79.484	-	8610	1034	24.156	1830	286	4.578	32	153	0.313
Average				0	734	77.115	1819	170.53405		1320	114.876		9700	948	194.923	6441	236	133.010	45	300	20.855

Table 3.7: PCMCA computational results for the MILP models *MCF, Compact-U^{st}, Compact-U^t, U^{st}, U^t* and *Set-Based* for SOPLIB instances.

Instance				Model														
				MCF			Compact-U^{st}		Compact-U^t		U^{st}			U^t			Set-Based		
Name	Size	$\rho(R)$	z^*	Nodes	Cuts	Time [s]	Nodes	Time [s]	Nodes	Time [s]	Nodes	Cuts	Time [s]	Nodes	Cuts	Time [s]	Nodes	Cuts	Time [s]
R.200.100.1	200	0.020	29	0	200	1.781	0	4.719	0	4.578	0	1	0.219	0	13	1.843	0	11	0.875
R.200.100.15	200	0.847	454	0	8600	713.657	1118	677.000	2608	338.312	382	3314	4034.859	269	729	29.531	177	2395	64.812
R.200.100.30	200	0.957	529	0	2600	8.375	28	19.922	28	17.562	59	142	54.828	28	68	3.656	10	77	0.875
R.200.100.60	200	0.991	6018	0	0	0.031	0	1.000	0	1.016	0	0	3.593	0	0	0.157	0	0	0.094
R.200.1000.1	200	0.020	887	0	200	3.625	0	9.562	0	9.875	0	0	0.203	0	2	0.625	0	3	0.656
R.200.1000.15	200	0.876	5891	0	6000	102.359	1156	52.266	997	44.719	132	731	329.313	74	328	9.360	87	557	7.860
R.200.1000.30	200	0.958	7653	0	1000	3.172	43	6.547	29	8.469	2	46	57.141	0	7	0.953	0	9	0.297
R.200.1000.60	200	0.989	6666	0	0	0.047	0	1.469	0	1.579	0	0	3.797	0	0	0.157	0	0	0.094
R.300.100.1	300	0.013	13	0	1200	84.782	0	10.515	0	3.360	0	0	0.500	0	22	5.313	0	14	2.250
R.300.100.15	300	0.905	575	0	16800	893.625	506	199.688	552	208.141	87859	119299	2220.656	476	1681	114.484	139	1111	55.734
R.300.100.30	300	0.970	756	0	1800	8.125	481	13.953	61	20.375	0	57	1.672	0	8	1.313	0	27	0.562
R.300.100.60	300	0.994	708	0	300	0.390	0	19.672	0	18.656	2	57	2.469	0	11	1.718	0	2	0.375
R.300.1000.1	300	0.013	715	0	900	297.563	0	58.562	0	62.375	0	69	10.546	0	69	10.343	0	8	2.515
R.300.1000.15	300	0.905	6660	0	10800	421.860	57	83.328	194	124.828	3304	4165	91.938	66	95	19.203	73	819	16.531
R.300.1000.30	300	0.965	8693	0	1800	4.110	0	7.328	0	10.718	0	11	1.531	0	1	1.016	0	6	0.453
R.300.1000.60	300	0.994	7678	0	600	0.672	0	10.672	0	11.109	0	4	23.234	0	0	0.469	0	2	0.297
R.400.100.1	400	0.010	6	0	0	0.688	0	18.093	0	19.515	0	1	0.391	0	7	1.142	2	45	9.750
R.400.100.15	400	0.927	699	0	22400	2291.765	1582	1895.359	564	1598.750	52858	105054	2021.813	499	1979	161.828	109	548	44.922
R.400.100.30	400	0.978	712	0	4800	8.172	0	21.110	0	24.875	0	58	10.156	0	2	8.078	0	14	2.031
R.400.100.60	400	0.996	557	0	0	0.234	0	10.265	0	11.031	0	2	0.219	0	1	0.181	0	0	0.328
R.400.1000.1	400	0.010	780	0	2400	1256.485	0	12.953	0	13.140	0	13	6.734	0	17	11.672	0	4	2.797
R.400.1000.15	400	0.930	7382	0	16000	569.484	1199	1265.079	608	1296.484	56018	170012	8935.188	328	153	75.516	91	362	24.000
R.400.1000.30	400	0.977	9368	0	3600	8.828	1979	174.156	2308	417.172	4797	5545	209.593	58	130	20.547	38	97	6.563
R.400.1000.60	400	0.995	7167	0	400	0.563	0	33.078	0	36.500	0	44	2.016	0	3	1.453	0	1	0.500
R.500.100.1	500	0.008	3	0	3000	1687.547	0	37.921	0	34.469	0	579	217.172	0	172	35.726	0	29	11.812
R.500.100.15	500	0.945	860	0	27000	1934.063	516	375.407	520	468.468	9879	8120	443.125	186	279	104.687	38	286	21.156
R.500.100.30	500	0.980	710	0	8000	197.953	0	68.609	0	61.734	11490	19359	696.922	9	16	25.969	15	51	3.562
R.500.100.60	500	0.996	566	0	500	1.078	0	43.265	0	43.234	0	0	0.687	0	0	0.625	0	1	0.844
R.500.1000.1	500	0.008	297	0	0	1.203	0	17.469	0	17.250	0	0	0.609	0	0	0.611	0	0	4.469
R.500.1000.15	500	0.940	8063	0	24000	1270.750	289	109.625	0	88.218	57	819	100.640	7	43	54.422	0	119	15.063
R.500.1000.30	500	0.981	9409	0	3500	4.922	0	30.516	0	32.954	0	28	11.141	0	2	7.985	0	11	3.125
R.500.1000.60	500	0.996	6163	0	500	1.360	0	36.984	0	40.718	0	0	0.671	0	0	0.634	0	1	0.875
R.600.100.1	600	0.007	1	0	3600	3397.079	0	81.797	0	81.532	0	858	659.156	0	1262	840.446	0	1455	733.375
R.600.100.15	600	0.950	568	0	19200	534.797	0	58.094	0	63.406	1	387	34.985	0	10	12.750	0	23	5.312
R.600.100.30	600	0.985	776	0	4200	46.375	1059	152.672	116	180.375	659	8656	298.109	0	13	15.578	0	24	2.375
R.600.100.60	600	0.997	538	0	0	0.610	0	24.453	0	24.672	0	0	0.359	0	0	0.265	0	0	0.906
R.600.1000.1	600	0.007	322	0	0	1.969	0	32.016	0	33.969	0	0	0.844	0	0	0.735	0	0	8.625
R.600.1000.15	600	0.945	9763	0	30600	1977.250	0	42.063	0	66.703	31	2092	159.515	4	20	56.547	0	69	12.766
R.600.1000.30	600	0.984	9497	0	4200	24.015	0	77.047	0	97.906	0	14	7.219	0	3	3.714	0	13	2.969
R.600.1000.60	600	0.997	6915	0	0	0.625	0	33.422	105	34.578	0	1	0.406	0	1	0.125	0	0	0.922
R.700.100.1	700	0.006	2	0	4900	4327.026	0	117.281	0	116.328	0	0	1.250	0	0	1.152	0	616	314.875
R.700.100.15	700	0.957	675	0	12600	206.344	0	176.047	0	86.688	0	106	41.000	0	7	12.766	0	23	6.875
R.700.100.30	700	0.987	590	0	700	1.656	0	61.203	0	74.266	0	0	3.984	0	0	2.813	0	1	1.250
R.700.100.60	700	0.997	383	0	0	0.703	0	46.437	0	45.469	0	0	0.500	0	0	0.435	0	0	1.422
R.700.1000.1	700	0.006	611	0	0	3.031	0	57.156	0	61.797	0	3	1.625	0	7	5.156	0	0	13.891
R.700.1000.15	700	0.956	2792	0	2100	7.094	0	28.609	0	35.828	0	3	1.500	0	1	5.156	0	4	1.875
R.700.1000.30	700	0.986	2658	0	0	0.516	0	20.078	0	23.500	0	0	0.360	0	0	0.259	0	0	0.828
R.700.1000.60	700	0.997	1913	0	0	0.640	0	55.750	0	60.719	0	0	0.515	0	0	0.315	0	0	1.375
Average				0	5229	464.771	209	133.130	181	128.707	4740	9368	431.352	42	149	34.780	16	184	29.494

Table 3.8: PCMCA computational results for the MILP models *MCF, Compact-U^{st}, Compact-U^{t}, U^{st}, U^{t}* and *Set-Based* for COMPILERS instances.

Instance				Model															
				MCF			Compact-U^{st}		Compact-U^{t}		U^{st}			U^{t}			Set-Based		
Name	Size	$\rho(R)$	z^*	Nodes	Cuts	Time [s]	Nodes	Time [s]	Nodes	Time [s]	Nodes	Cuts	Time [s]	Nodes	Cuts	Time [s]	Nodes	Cuts	Time [s]
gsm.153.124	126	0.970	185	0	378	0.375	0	1.610	0	1.125	0	180	0.578	0	26	0.297	3	53	0.140
gsm.444.350	353	0.990	1542	0	0	0.078	0	0.594	0	0.454	0	2	0.078	0	12	0.375	0	0	0.094
gsm.462.77	79	0.840	292	0	474	0.875	0	2.109	0	2.234	17	79	4.047	0	31	0.296	0	14	0.031
jpeg.1483.25	27	0.484	71	0	54	0.172	0	1.156	8	1.062	43	197	0.266	16	55	0.125	4	34	0.047
jpeg.3184.107	109	0.887	411	0	981	1.953	0	1.391	2616	5.765	24	117	16.844	37	83	0.922	0	32	0.093
jpeg.3195.85	87	0.740	13	0	1392	6.328	47	87.672	1	55.156	4041	47994	1366.985	155	4003	120.359	5674	16979	897.312
jpeg.3198.93	95	0.752	140	0	27343	15.531	0	17.390	0	28.203	2204	6649	529.781	48	1921	34.329	401	1686	9.704
jpeg.3203.135	137	0.897	507	0	1918	4.109	495	19.046	485	13.141	31	196	56.703	35	132	2.141	7	41	0.125
jpeg.3740.15	17	0.257	33	0	34	0.125	57	3.922	40	4.079	231	185	0.234	391	78	0.188	0	17	0.031
jpeg.4154.36	38	0.633	74	0	228	0.734	321	8.344	251	3.984	1462	364	2.500	692	156	0.812	0	43	0.063
jpeg.4753.54	56	0.769	146	0	224	0.469	0	4.391	0	2.641	11	192	2.984	6	74	0.375	6	59	0.109
susan.248.197	199	0.939	688	0	2587	8.110	1984	27.968	1984	25.031	22	178	106.672	9	79	2.922	0	21	0.125
susan.260.158	160	0.916	472	0	2400	4.860	1534	180.062	6909	55.953	570	461	123.594	261	111	4.578	0	33	0.141
susan.343.182	184	0.936	468	0	2760	6.829	516	44.016	3567	72.703	776	896	474.391	318	399	12.812	19	89	0.359
typeset.10192.123	125	0.744	241	0	1500	10.359	4674	2267.172	2879	1437.594	5565	1134	297.859	4537	644	40.766	0	93	0.500
typeset.10835.26	28	0.349	60	0	112	0.407	0	0.079	0	0.078	0	8	0.063	0	9	0.047	0	14	0.031
typeset.12395.43	45	0.518	125	0	270	1.000	0	1.250	29	1.594	10	64	0.437	0	28	0.125	0	27	0.078
typeset.15087.23	25	0.557	89	0	50	0.250	23	0.985	16	2.250	32	148	0.297	67	51	0.109	0	24	0.047
typeset.15577.36	38	0.555	93	0	76	0.204	0	0.516	0	0.468	0	7	0.031	0	6	0.062	0	4	0.015
typeset.16000.68	70	0.658	67	0	490	1.219	16981	3872.728	6317	2532.500	0	787	21.891	0	425	17.725	144	1316	7.172
typeset.1723.25	27	0.245	54	0	216	1.375	4237	95.156	82554	65.891	7660	781	4.094	9000	283	7.094	21	99	0.110
typeset.19972.246	248	0.993	979	0	0	0.046	0	0.234	0	0.250	0	14	0.110	0	7	0.069	0	0	0.062
typeset.4391.240	242	0.981	837	0	2178	2.735	0	5.156	0	6.359	46	1291	6.250	0	95	3.782	0	18	0.094
typeset.4597.45	47	0.493	133	0	94	0.235	0	0.234	0	0.297	0	16	0.437	0	16	0.079	0	7	0.031
typeset.4724.433	435	0.995	1819	0	3045	3.578	10	17.109	22	23.704	0	374	4.000	0	68	1.823	0	8	0.172
typeset.5797.33	35	0.748	93	0	35	0.094	0	0.407	0	0.297	0	85	0.234	0	37	0.125	0	9	0.032
typeset.5881.246	248	0.986	979	0	496	0.484	1289	34.547	1258	38.062	191	184	356.218	394	137	7.031	0	52	0.343
Average				0	1827	2.686	1191	247.972	4035	162.255	849	2318	125.095	591	332	9.606	233	769	33.965

3.6.2 The B&B Algorithm

In this section we discuss the computational results of the B&B algorithm, and compare it with solving the *Set-Based Model* (see section 3.4.4) with an MILP solver.

3.6.2.1 Lagrangian Relaxation

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 [5]. 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.

Figure 3.12 compares the value of objective function (3.56) 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.

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 (3.56), in addition to the MCA being computed at each iteration, and thus increasing the

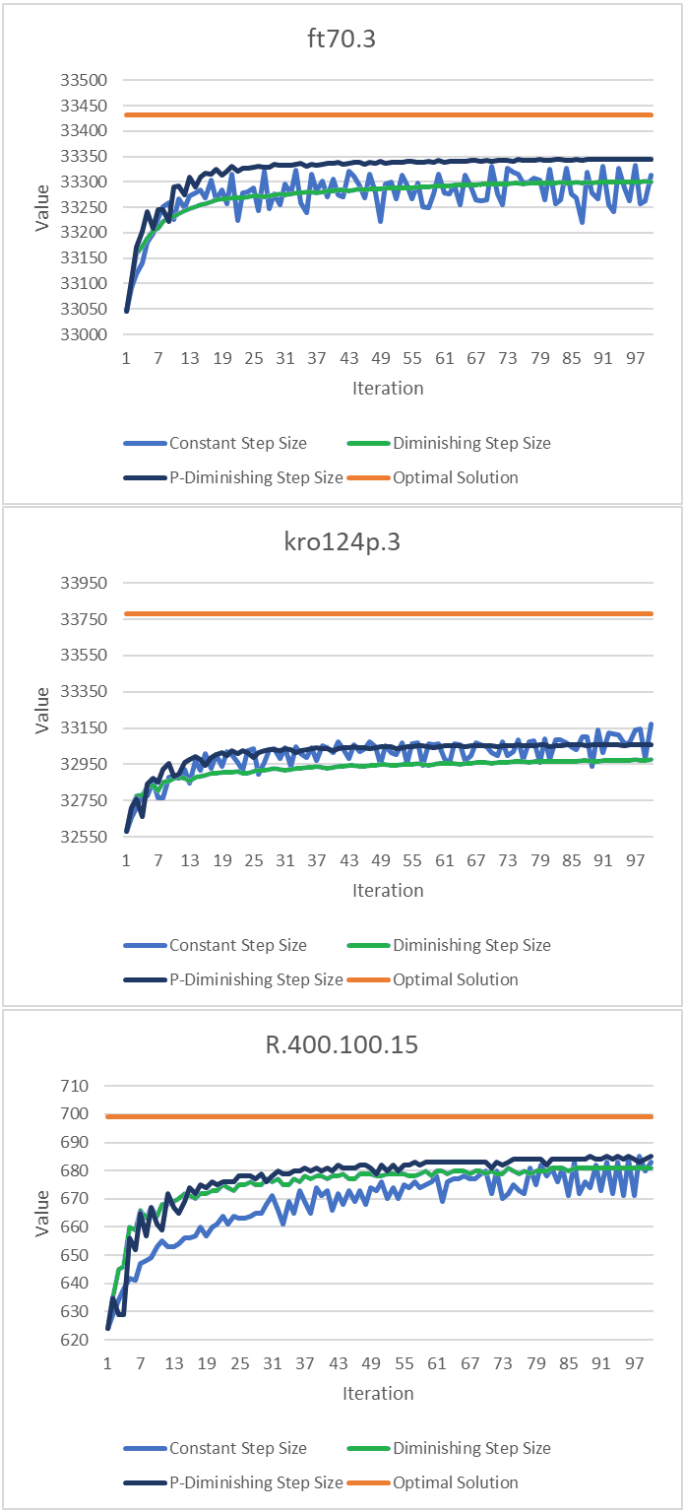

Figure 3.12: Comparing the value of the objective function at each iteration using a constant step size $\alpha_k = 0.1$, diminishing step size $\alpha_k = \frac{1}{k}$, and p-diminishing step size $\alpha_k = \frac{1}{p}$.

computation time per node. Figure 3.12 shows also the value of the lower bound at the root 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 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 Figure 3.13. 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.

3.6.2.2 Overall Results

In this section, 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 section 3, that are obtained by solving the MILP model (*Set-Based model*) introduced in section 3.4.4. The results of the others methods/models discussed in chapter 3 are not compared against, as they are generally dominated with computation times that are 906.2% and 131.6% higher on average.

Tables 3.9-3.11 report the results for the three benchmark sets. The tables compare the performance of the current state-of-the-art solver/model described in section 3.4.4, with the B&B algorithm introduced in section 3.5.2 using different step size rules. In all the tables, column *Name* reports the name of the instance, column *Size* reports the number of vertices in the instance graph, and column $\rho(R)$ reports the density of arcs in the set of precedence relationships R, and is computed as $\frac{2 \cdot |R|}{|V|(|V|-1)}$. Column z^* reports the value of the optimal solution of each instance. For

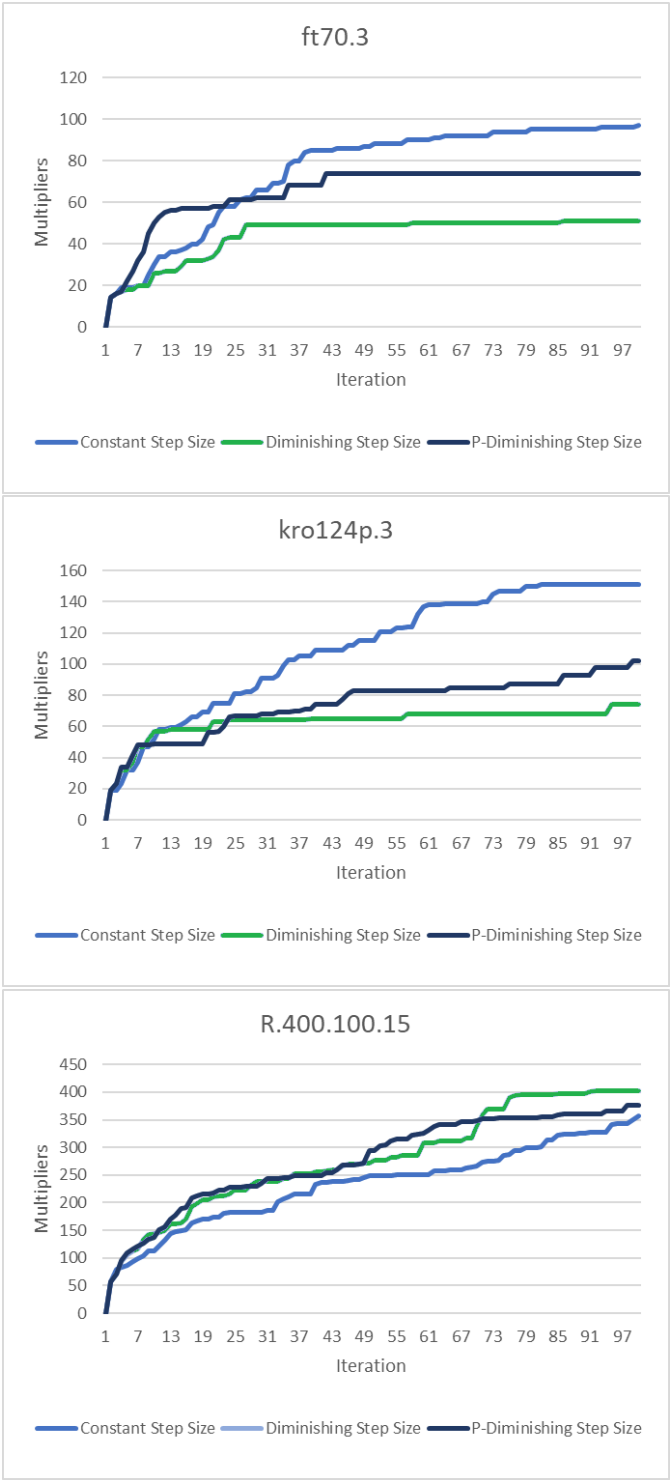

Figure 3.13: Comparing the increase in the number Lagrangian multipliers for the root node at each iteration using a constant step size $\alpha_k = 0.1$, diminishing step size $\alpha_k = \frac{1}{k}$, and p-diminishing step size $\alpha_k = \frac{1}{p}$.

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. The maximum number of iterations $M = 10$ is used for the B&B algorithm.

An overview of the results show that the B&B algorithm optimally solves the three benchmark sets. Solving the MILP has an average solution time of 27.5 seconds, 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, 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 $\alpha_k = \frac{1}{p}$.

The bypass rule described in Section 3.5.5 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 hour 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 seconds, 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 seconds, 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 takes on average 27.7 seconds, 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 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.

Table 3.9: PCMCA computational results comparing solving the MILP model *Set-Based* and the B&B algorithm for TSPLIB instances.

Instance				Set-Based Model		B&B \w $\alpha_k = 0.1$		B&B \w $\alpha_k = \frac{1}{k}$		B&B \w $\alpha_k = \frac{1}{p}$	
Name	Size	$\rho(R)$	z^*	Nodes	Time [s]	Nodes	Time [s]	Nodes	Time [s]	Nodes	Time [s]
br17.10	18	0.314	25	0	0.015	1	0.004	0	0.001	0	0.004
br17.12	18	0.359	25	0	0.016	1	0.004	0	0.006	0	0.006
ESC07	9	0.611	1531	0	0.031	0	0.000	0	0.001	0	0.000
ESC11	13	0.359	1752	0	0.031	0	0.000	0	0.001	0	0.000
ESC12	14	0.396	1138	0	0.016	0	0.001	0	0.001	0	0.001
ESC25	27	0.177	1041	0	0.063	0	0.001	0	0.002	0	0.001
ESC47	49	0.108	703	0	0.547	1	0.013	1	0.009	0	0.018
ESC63	65	0.173	56	0	0.218	4	0.095	3	0.036	1	0.052
ESC78	80	0.139	502	0	0.047	0	0.019	0	0.008	0	0.012
ft53.1	54	0.082	3917	5	0.375	6	0.051	3	0.028	1	0.021
ft53.2	54	0.094	3978	55	0.547	19	0.158	11	0.128	8	0.107
ft53.3	54	0.225	4242	0	0.453	7	0.116	39	0.384	11	0.102
ft53.4	54	0.604	4882	0	0.047	9	0.064	7	0.096	6	0.067
ft70.1	71	0.036	32846	0	2.750	2	0.043	2	0.043	1	0.039
ft70.2	71	0.075	32930	0	2.719	4	0.143	4	0.099	5	0.122
ft70.3	71	0.142	33431	145	38.250	170	6.622	75	2.607	92	2.789
ft70.4	71	0.589	35179	369	6.281	45	0.662	136	2.371	14	0.395
rbg048a	50	0.444	204	0	0.031	0	0.010	1	0.007	0	0.005
rbg050c	52	0.459	191	0	0.047	3	0.034	3	0.031	1	0.011
rbg109	111	0.909	256	0	0.094	0	0.025	0	0.019	0	0.017
rbg150a	152	0.927	373	1	0.219	0	0.041	0	0.028	0	0.031
rbg174a	176	0.929	365	1	0.313	1	0.083	0	0.044	0	0.046
rbg253a	255	0.948	375	0	1.125	0	0.162	0	0.162	11	0.636
rbg323a	325	0.928	754	0	1.047	0	0.265	0	0.204	0	0.235
rbg341a	343	0.937	610	0	3.031	13	1.706	7	0.634	5	0.579
rbg358a	360	0.886	595	0	5.812	1	0.595	0	0.259	0	0.262
rbg378a	380	0.894	559	36	19.047	597	77.582	12	3.342	223	32.326
kro124p.1	101	0.046	32597	0	1.782	2	0.064	2	0.064	2	0.062
kro124p.2	101	0.053	32851	27	3.281	151	5.268	224	8.884	137	5.440
kro124p.3	101	0.092	33779	98	7.469	2000	308.242	2000	294.649	2000	279.382
kro124p.4	101	0.496	37124	0	1.672	16	0.669	66	3.399	12	0.531
p43.1	44	0.101	2720	128	1.765	0	0.002	0	0.003	0	0.003
p43.2	44	0.126	2720	237	4.359	0	0.002	0	0.004	0	0.039
p43.3	44	0.191	2720	134	1.437	1	0.056	1	0.066	1	0.065
p43.4	44	0.164	2820	353	2.797	2	0.041	0	0.011	0	0.009
prob.100	100	0.048	650	4	743.969	125	5.565	48	2.377	37	1.560
prob.42	42	0.116	143	0	0.032	0	0.002	0	0.002	0	0.007
ry48p.1	49	0.091	13095	54	0.609	132	1.289	159	1.386	107	1.073
ry48p.2	49	0.103	13103	0	0.235	15	0.197	13	0.138	10	0.099
ry48p.3	49	0.193	13886	146	2.156	2000	24.712	2000	20.654	1961	18.655
ry48p.4	49	0.588	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

Table 3.10: PCMCA computational results comparing solving the MILP model *Set-Based* and the B&B algorithm for SOPLIB instances.

Instance				Set-Based Model		B&B \w $\alpha_k = 0.1$		B&B \w $\alpha_k = \frac{1}{k}$		B&B \w $\alpha_k = \frac{1}{p}$	
Name	Size	$\rho(R)$	z^*	Nodes	Time [s]	Nodes	Time [s]	Nodes	Time [s]	Nodes	Time [s]
R.200.100.1	200	0.020	29	0	0.875	0	0.040	0	0.031	0	0.030
R.200.100.15	200	0.847	454	177	64.812	1585	61.350	719	20.733	739	21.166
R.200.100.30	200	0.957	529	10	0.875	8	0.181	21	0.348	10	0.200
R.200.100.60	200	0.991	6018	0	0.094	0	0.049	0	0.048	0	0.056
R.200.1000.1	200	0.020	887	0	0.656	0	0.031	0	0.028	0	0.029
R.200.1000.15	200	0.876	5891	87	7.860	50	1.006	71	1.267	29	0.653
R.200.1000.30	200	0.958	7653	0	0.297	7	0.178	13	0.230	6	0.147
R.200.1000.60	200	0.989	6666	0	0.094	0	0.064	0	0.046	0	0.051
R.300.100.1	300	0.013	13	0	2.250	0	0.077	0	0.064	0	0.097
R.300.100.15	300	0.905	575	139	55.734	517	22.353	94	4.739	190	9.592
R.300.100.30	300	0.970	756	0	0.562	3	0.236	1	0.165	1	0.168
R.300.100.60	300	0.994	708	0	0.375	1	0.196	1	0.184	0	0.159
R.300.1000.1	300	0.013	715	0	2.515	0	0.067	0	0.062	0	0.062
R.300.1000.15	300	0.905	6660	73	16.531	13	0.695	52	2.366	9	0.474
R.300.1000.30	300	0.965	8693	0	0.453	14	0.648	10	0.299	3	0.205
R.300.1000.60	300	0.994	7678	0	0.297	2	0.249	2	0.193	2	0.211
R.400.100.1	400	0.010	6	2	9.750	0	0.141	0	0.122	0	0.115
R.400.100.15	400	0.927	699	109	44.922	2000	425.474	2000	268.596	2000	291.932
R.400.100.30	400	0.978	712	0	2.031	2	0.481	3	0.397	5	0.633
R.400.100.60	400	0.996	557	0	0.328	0	0.379	0	0.285	0	0.322
R.400.1000.1	400	0.010	780	0	2.797	0	0.130	0	0.106	0	0.119
R.400.1000.15	400	0.930	7382	91	24.000	702	41.821	199	10.402	168	9.102
R.400.1000.30	400	0.977	9368	38	6.563	27	2.376	51	4.411	31	2.510
R.400.1000.60	400	0.995	7167	0	0.500	0	0.351	0	0.354	0	0.348
R.500.100.1	500	0.008	3	0	11.812	0	0.194	0	0.179	0	0.189
R.500.100.15	500	0.945	860	38	21.156	31	2.873	15	1.442	5	0.896
R.500.100.30	500	0.980	710	15	3.562	7	1.278	11	1.714	13	1.726
R.500.100.60	500	0.996	566	0	0.844	0	0.685	0	0.639	0	0.632
R.500.1000.1	500	0.008	297	0	4.469	0	0.214	0	0.177	0	0.192
R.500.1000.15	500	0.940	8063	0	15.063	5	1.256	6	0.860	6	0.823
R.500.1000.30	500	0.981	9409	0	3.125	9	1.217	17	1.749	6	0.891
R.500.1000.60	500	0.996	6163	0	0.875	0	0.773	0	0.613	0	0.632
R.600.100.1	600	0.007	1	0	733.375	0	0.314	0	0.248	0	0.303
R.600.100.15	600	0.950	568	0	5.312	0	1.302	1	0.973	0	0.983
R.600.100.30	600	0.985	776	0	2.375	5	2.080	3	1.207	2	1.132
R.600.100.60	600	0.997	538	0	0.906	0	0.225	0	0.981	0	1.076
R.600.1000.1	600	0.007	322	0	8.625	0	0.303	0	0.260	0	0.299
R.600.1000.15	600	0.945	9763	0	12.766	4	1.155	5	1.238	1	1.143
R.600.1000.30	600	0.984	9497	0	2.969	3	1.184	7	1.375	2	1.228
R.600.1000.60	600	0.997	6915	0	0.922	0	0.336	0	0.987	0	1.089
R.700.100.1	700	0.006	2	0	314.875	0	0.446	0	0.336	0	0.380
R.700.100.15	700	0.957	675	0	6.875	8	2.587	1	1.544	2	1.734
R.700.100.30	700	0.987	590	0	1.250	0	0.975	0	0.973	0	0.961
R.700.100.60	700	0.997	383	0	1.422	0	0.994	0	0.982	0	0.991
R.700.1000.1	700	0.006	611	0	13.891	0	0.440	0	0.358	0	0.402
R.700.1000.15	700	0.956	2792	0	1.875	0	0.825	0	0.889	0	0.881
R.700.1000.30	700	0.986	2658	0	0.828	0	0.926	0	0.952	0	0.911
R.700.1000.60	700	0.997	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

Table 3.11: PCMCA computational results comparing solving the MILP model *Set-Based* and the B&B algorithm for COMPILERS instances.

Instance				Set-Based Model		B&B \w $\alpha_k = 0.1$		B&B \w $\alpha_k = \frac{1}{k}$		B&B \w $\alpha_k = \frac{1}{p}$	
Name	Size	$\rho(R)$	z^*	Nodes	Time [s]	Nodes	Time [s]	Nodes	Time [s]	Nodes	Time [s]
gsm.153.124	126	0.970	185	3	0.140	0	0.019	0	0.012	0	0.016
gsm.444.350	353	0.990	1542	0	0.094	0	0.246	0	0.204	0	0.198
gsm.462.77	79	0.840	292	0	0.031	0	0.037	1	0.052	0	0.039
jpeg.1483.25	27	0.484	71	4	0.047	0	0.042	0	0.007	0	0.003
jpeg.3184.107	109	0.887	411	0	0.093	0	0.057	1	0.036	4	0.040
jpeg.3195.85	87	0.740	13	5674	897.312	9	2.992	6	2.940	6	3.062
jpeg.3198.93	95	0.752	140	401	9.704	27	0.334	68	0.551	162	2.742
jpeg.3203.135	137	0.897	507	7	0.125	6	0.075	1	0.028	0	0.022
jpeg.3740.15	17	0.257	33	0	0.031	33	0.099	31	0.098	5	0.019
jpeg.4154.36	38	0.633	74	0	0.063	0	0.014	4	0.036	65	0.526
jpeg.4753.54	56	0.769	146	6	0.109	0	0.010	0	0.011	0	0.004
susan.248.197	199	0.939	588	0	0.125	1	0.063	0	0.048	0	0.053
susan.260.158	160	0.916	472	0	0.141	4	0.142	8	0.178	7	0.137
susan.343.182	184	0.936	468	19	0.359	12	0.355	8	0.203	13	0.365
typeset.10192.123	125	0.744	241	0	0.500	42	0.909	103	1.371	96	1.159
typeset.10835.26	28	0.349	60	0	0.031	0	0.002	0	0.001	0	0.001
typeset.12395.43	45	0.518	125	0	0.078	0	0.004	0	0.003	0	0.003
typeset.15087.23	25	0.557	89	0	0.047	4	0.013	3	0.022	2	0.012
typeset.15577.36	38	0.555	93	0	0.015	0	0.002	0	0.002	0	0.017
typeset.16000.68	70	0.658	67	144	7.172	424	5.393	3	1.123	9	0.151
typeset.1723.25	27	0.245	54	21	0.110	127	0.888	117	0.558	106	0.528
typeset.19972.246	248	0.993	979	0	0.062	0	0.093	0	0.078	0	0.083
typeset.4391.240	242	0.981	837	0	0.094	2	0.105	1	0.083	1	0.087
typeset.4597.45	47	0.493	133	0	0.031	0	0.009	0	0.051	0	0.004
typeset.4724.433	435	0.995	1819	0	0.172	0	0.038	0	0.372	0	0.405
typeset.5797.33	35	0.748	93	0	0.032	0	0.037	0	0.003	0	0.009
typeset.5881.246	248	0.986	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

3.7 Conclusions

In this chapter we introduced a new variation on the *Minimum-Cost Arborescence problem* named the *Precedence-Constrained Minimu-Cost Arborescence problem*, and presented a proof that the problem belongs to the $\mathcal{NP}$-hard complexity class through a reduction to the 3-SAT problem.

We proposed several MILP models for the PCMCA problem, that are either exponential or polynomial in size. The computational results has shown that the *Set-Based* model generally outperforms the rest of the models proposed. However, in certain cases the U^t model is more effective than the *Set-Based* model on instances with dense precedence graphs. The computational results has shown that the two models U^t and *Set-Based* are able to optimally solve all 116 benchmark instances, however the model *Set-Based* is 57.1% faster on average.

A *Branch-and-Bound* algorithm for the PCMCA problem is proposed that is based on a *Lagrangian relaxation* of the *Set-Based* model. A set of pruning techniques are proposed where some of them takes advantage of problem specific structures in order to reduced the size of the explored solution space. The computational results has shown that the proposed B&B algorithm is 94% faster on average at solving the instances, compared to solving the original model with a MILP solver.

Chapter 4

The Precedence-Constrained Minimum-Cost Arborescence Problem with Waiting Times

The aim of this chapter is to introduce the *Precedence-Constrained Minimum-Cost Arborescence Problem with Waiting-Times* (PCMCA-WT), that is an extension to the PCMCA problem introduced in chapter 3. In section 4.1 we define the PCMCA-WT problem. In section 4.2 we present a proof of complexity to the PCMCA-WT problem. In section 4.3 we propose a formulation to model waiting-times in an arborescence. In section 4.4 we propose several MILP models for the PCMCA-WT, while in section 4.5 we propose two constraint programming (CP) models for the PCMCA-WT. Finally, section 4.6 discusses computational results and compares the different models presented, while conclusions are drawn in section 4.7. The work presented in this chapter has appeared in [12, 22].

4.1 Problem Definition

The Precedence-Constrained Minimum-Cost Arborescence problem with Waiting-Times [12] can be described by extending the definition of the PCMCA problem (see section 3.1) as follows. Assume there is a flow which starts at the root vertex r at time 0, and traverses each path of the arborescence. The cost c_{ij} of an arc $(i, j) \in A$ represents the time required to traverse that arc. Let d_j be the time at which the flow enters vertex $j \in V$. For any $(s, t) \in R$, $d_t \geq d_s$, which means that the flow must enter vertex t at the same time step or after entering vertex s, but can stop at any vertex and wait. Let w_j be the waiting time before the flow enters vertex j required to respect the aforementioned constraint. The objective of the problem is to find an arborescence T that has a minimum total cost plus total waiting time, where the flow never enters t earlier than entering s for all $(s, t) \in R$. Note that any feasible solution does not contain a unique path covering s and t, that visits t before visiting s, otherwise the flow enters t before entering s, therefore any feasible PCMCA-WT solution is also a feasible PCMCA solution.

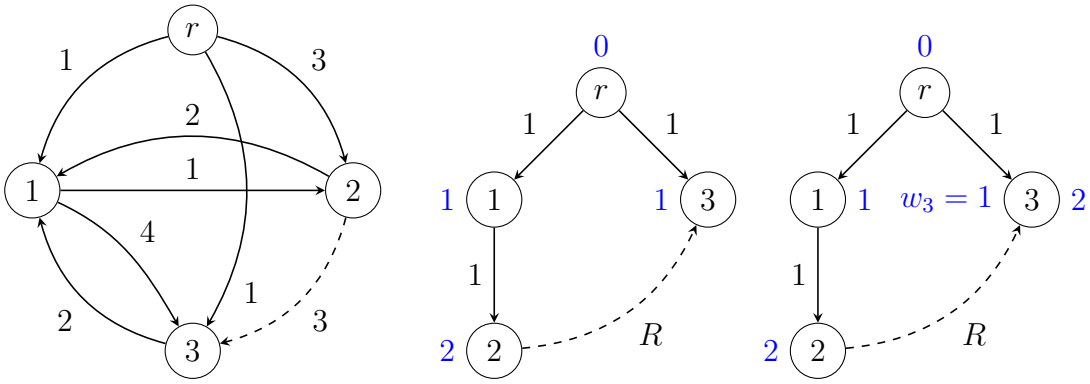

Figure 4.1: Comparing an instance solved as a PCMCA, and solved as a PCMCA-WT.

Figure 4.1 presents an example that shows the difference between the PCMCA problem and the PCMCA-WT problem. The graph on the left shows the instance graph with the precedence relationship $(2, 3) \in R$ represented as a dashed arrow.

The graphs in the middle and on the left respectively show an optimal PCMCA solution and an optimal PCMCA-WT solution, where next to each vertex we have its corresponding d_t value. The two solutions depicted are valid solutions for the PCMCA problem, since they both satisfy the precedence constraints, that is t never precede s on the same directed path for all $(s,t) \in R$. The solution in the middle shows the optimal PCMCA solution with a total cost of 3 (sum of all the arcs). We can see that the solution in the middle is not a feasible PCMCA-WT solution since $(2,3) \in R$ but $d_3 < d_2$. The solution on the right shows an optimal PCMCA-WT solution with a cost of 4 (sum of all the arcs plus waiting time at each vertex). The solution results in a waiting time of 1 at vertex 3, since the time from r to 2 is 2, and the time from r to 3 is 1.

4.2 Computational Complexity

In this section we present a proof that the PCMCA-WT problem is $\mathcal{NP}$-hard by a reduction from the *Rectilinear Steiner Arborescence problem* (RSA) [78].

The RSA problem is an $\mathcal{NP}$-hard problem [78] formally defined as follows. Let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of points in the first quadrant of the Cartesian plane, where $p_i = (x_i, y_i)$ with $x_i, y_i \geq 0$, and $p_1 = (0,0)$. A complete grid can be created, where the points in P are on the intersections of vertical and horizontal lines. A set S of Steiner vertices can be added, corresponding to the $O(|P|^2)$ intersection points not overlapping with the points in P. The arcs of the problem are the right-directed horizontal segments and the up-directed vertical segments between two adjacent points of the grid $P \cup S$. The cost associated with each arc (p_i, p_j) is defined as $|x_i - x_j| + |y_i - y_j|$.

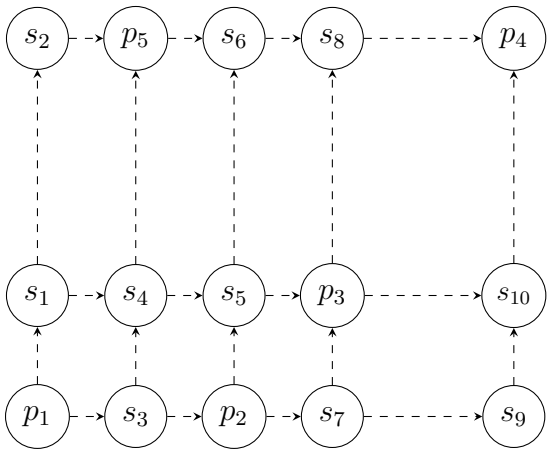

Figure 4.2: Example of an RSA instance with 5 points and 10 Steiner vertices.

Figure 4.2 shows an example of an RSA instance with 5 points, and the relative Steiner vertices, while the dashed lines represent the arcs of the instance. Given a positive value k, the decision version of the RSA problem consists in deciding whether there is an arborescence with total length not greater than k such that the arborescence is rooted at p_1 and it contains a unique path from p_1 to p_i for all $i \in \{1, 2, \ldots, n\}$. Note that the length of each path from p_1 to p_i is $x_i + y_i$ by construction.

Theorem 4.1. *The PCMCA-WT is $\mathcal{NP}$-hard*

Proof. By a reduction from the decision version of the RSA problem: we construct a graph $G = (V, A)$ and a set R of precedence constraints such that there exist a PCMCA-WT solution of cost at most k if and only if a RSA of cost at most k exists. Given an instance of the RSA problem with a set of points P and a set of Steiner points S, consider the PCMCA-WT instance defined as follows:

$$
\begin{aligned}
V &= P \cup S \\
A' &= \{(i, j) : j \text{ is immediately on the top of } i \text{ in the grid, or } j \text{ is immediately} \\
&\qquad \text{on the right of } i \text{ in the grid}\}
\end{aligned}
$$

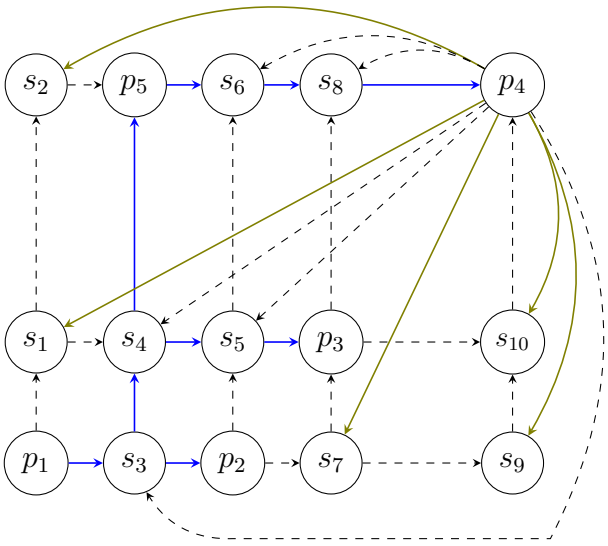

Figure 4.3: The PCMCA-WT instance associated with the RSA instance depicted in Figure 4.2. A RSA solution of minimum cost is given by the blue arcs. The green arcs have cost 0 and, together with the blue ones, form an optimal PCMCA-WT solution.

$$
\begin{aligned}
A &= A' \cup \{(P_{FAR}, s_i), s_i \in S\}, \text{ with } P_{FAR} \in \underset{p_i \in P}{\operatorname{argmax}}\{x_i + y_i\} \\
R &= \{(p, P_{FAR}) : p \in P \setminus \{P_{FAR}\}\} \\
c_{ij} &= (x_j - x_i) + (y_j - y_i) \text{ for } (i,j) \in A' \\
c_{P_{FAR},s_i} &= 0 \text{ for } s_i \in S \\
r &= p_1
\end{aligned}
$$

If the instance of RSA has a solution of cost k, then a solution of cost k for the instance of PCMCA-WT can be obtained. Starting from the solution of the RSA problem, it is possible to complete the solution of the associated PCMCA-WT problem by adding 0-cost arcs (green arcs) to connect the node P_{FAR} to the Steiner nodes not used in the RSA solution. The solution of an RSA instance and a solution of the associated PCMCA-WT problem are depicted in Figure 4.3.

Conversely, assume that there is a feasible solution of PCMCA-WT with cost at

most k. Without loss of generality suppose that such a solution is optimal. Note that a path starting at P_{FAR} and passing through a vertex in P cannot exist due to the precedence constraints. Besides, every leaf of the arborescence that is in S must have P_{FAR} as parent; otherwise, making P_{FAR} its parent would reduce the cost. Therefore, removing all the leaves of the PCMCA-WT arborescence connected through P_{FAR} results in a tree that uses only arcs in A' and whose leaves are all in P. It follows that the resulting tree is a feasible solution for the RSA. □

4.3 Flow-Precedence Constraints

Flow-Precedence Constraints are a set of constraints which enforce that for any precedence relationship $(s,t) \in R$ the flow must never enter vertex t before entering vertex s. Such a constraint can be enforced by either enforcing that if both s and t belong to the same directed path, then s must precede t on that path. However, if s and t belong to two disjoint paths, then a waiting time must be enforced on t, that is equal to the the difference between the time at which the flow reaches t and the time at which the flow enters s. Let d_j be the time at which the flow enters vertex $j \in V$, and let w_j be the waiting time before the flow enters vertex j. Let x_{ij} be a variable associated with every arc $(i,j) \in A$ such that $x_{ij} = 1$ if $(i,j) \in T$ and 0 otherwise. The aforementioned constraints can be modeled as the following sets of inequalities.

$$d_r = 0 \tag{4.1}$$

$$w_r = 0 \tag{4.2}$$

$$d_j \geq d_i - M + (M + c_{ij})x_{ij} \qquad \forall (i,j) \in A \tag{4.3}$$

$$w_j \geq d_j - d_i - M + (M - c_{ij})x_{ij} \qquad \forall (i,j) \in A \tag{4.4}$$

$$d_t \geq d_s \qquad \forall (s,t) \in R \tag{4.5}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A \tag{4.6}$$

$$d_i, w_i \geq 0 \qquad \forall i \in V \tag{4.7}$$

Constraint (4.1) sets the time at which the flow enters the root r to 0. Constraint (4.2) set the waiting time at the root vertex r to 0. Constraints (4.3) impose that when arc $(i,j) \in A$ is selected to be part of the arborescence, then the time at which the flow enters vertex j is greater than or equal to the time at which the flow enters vertex i plus c_{ij}. Constraints (4.4) enforce that the waiting time at each vertex j is greater than or equal to the difference between the time at which the flow enters vertex j and the time at which the flow enters vertex i plus c_{ij}, where i is the parent of j in the arborescence. Constraints (4.5) enforce that the time at which the flow enters vertex t must be greater than or equal to the time at which the flow enters vertex s, for all $(s,t) \in R$. Finally, constraints (4.6) and (4.7) define the domain of the variables.

The set on inequalities (4.3) and (4.4) uses a *Big-M* [14] in order to turn the constraint off when vertex i is not the parent of vertex j in the current solution (i.e. $x_{ij} = 0$). The value of M is an upper bound on the value of the optimal solution, which is a sufficiently large integer. However, having the value of M as close as possible to the value of the integer solution would affect how hard or easy it is to solve the model.

One way to approximate the value of M is to compute a simple directed path rooted at r which covers all the vertices of V, and does not contain a (t,s)-path for all $(s,t) \in R$. Such a path can be efficiently constructed using a *Nearest Neighbor* algorithm as described in Algorithm 6.

Algorithm 6 Nearest Neighbor Algorithm for Computing *Big-M*

```
1: procedure ComputeBig-M(G, r)
2:     P = [r]
3:     M = 0
4:     while |P| < |V| do
5:         Let i be the last element in P
6:         Let N be an empty set of vertices.
7:         for j ∈ V do
8:             if NotViolating(P, j) then
9:                 N = N ∪ {j}
10:            end if
11:        end for
12:        Sort N according to the distance of each vertex from i
13:        Let j be the first element in N
14:        P = P ∪ [j]
15:        M = M + c_ij
16:    end while
17: end procedure
```

The second line of the algorithm initializes a list that contains only the root vertex r, while the third line initializes the value of M to zero. The while loop on line 4 stops executing when the path P contains all the vertices of the graph G. Lines 6-11 create a list of vertices ordered by the their distance from the last vertex in the path P (i.e. vertex i). A vertex j is added to N if and only if adding vertex j to P does not create a violating path. The function *NotViolating* returns true if vertex $s \in P$ for all $(s, j) \in R$. Finally, lines 12-14 append the nearest neighbor of vertex i to P and increments the value of M by c_{ij}.

4.4 Mixed Integer Linear Programming Models

In this section we introduce several MILP models for the PCMCA-WT problem that extend the models proposed for the PCMCA problem by adding the *Flow-Precedence*

Constraints to them.

4.4.1 Multicommodity Flow Model

In this section we describe a MILP model for the PCMCA-WT that extends the Multicommodity flow model for the PCMCA problem previously described in section 3.4.1.

$$\text{minimize} \sum_{(i,j)\in A} c_{ij}x_{ij} + \sum_{j\in V} w_j \tag{4.8}$$

$$\text{subject to} \sum_{(i,j)\in A} x_{ij} = 1 \qquad \forall j \in V\backslash\{r\} \tag{4.9}$$

$$\sum_{\substack{(i,j)\in A:\\(k,j)\notin R}} y_{ij}^k - \sum_{\substack{(j,i)\in A:\\(k,j)\notin R}} y_{ji}^k = \begin{cases} 1 \text{ if } i = r \\ -1 \text{ if } i = k \\ 0 \text{ otherwise} \end{cases} \qquad \begin{array}{r} \forall k \in V\backslash\{r\}, \\ \forall i \in V : (k,i) \notin R \end{array} \tag{4.10}$$

$$d_r = 0 \tag{4.11}$$

$$w_r = 0 \tag{4.12}$$

$$d_j \geq d_i - M + (M + c_{ij})x_{ij} \qquad \forall (i,j) \in A \tag{4.13}$$

$$w_j \geq d_j - d_i - M + (M - c_{ij})x_{ij} \qquad \forall (i,j) \in A \tag{4.14}$$

$$d_t \geq d_s \qquad \forall (s,t) \in R \tag{4.15}$$

$$y_{ij}^k \leq x_{ij} \qquad \forall k \in V\backslash\{r\}, (i,j) \in A \tag{4.16}$$

$$y_{ij}^k \in \{0,1\} \qquad \forall k \in V\backslash\{r\}, (i,j) \in A \tag{4.17}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A \tag{4.18}$$

$$d_i, w_i \geq 0 \qquad \forall i \in V\text{‘} \tag{4.19}$$

Constraints (4.10) impose the first property of an arborescence namely that every vertex $v \in V\backslash\{r\}$ must have a single parent. Constraints (4.10) are the Multicommodity flow constraints: every vertex $k \in V$ must be reachable from the root, and any path from r to k must not pass through the successors of k in the set of precedence relationships R (otherwise this would violate a precedence relation). Constraint (4.11) sets the time at which the flow enters the root to 0. Constraints (4.12) set the waiting time at the root r to 0. Constraints (4.13) impose that when arc $(i, j) \in A$ is selected to be part of the arborescence, then the time at which the flow enters vertex j is greater than or equal to the time at which the flow enters vertex i plus c_{ij}. Constraints (4.14) enforce that the waiting time at each vertex j is greater than or equal to the difference between the time at which the flow enters vertex j and the time at which the flow enters vertex i plus c_{ij}, where i is the parent of j in the arborescence. Constraints (4.15) enforce that the time at which the flow enters vertex t must be greater than or equal to the time at which the flow enters vertex s, for all $(s, t) \in R$. Finally, constraints (4.16)-(4.19) define the domain of the variables. Constraints (4.10) are dynamically added to the model using the same separation procedure described by algorithm 2 in section 3.4.1.

4.4.2 Path-Based Models

In this section we describe two MILP models for the PCMCA-WT that are polynomial in size, but use a smaller number of variables and constraints compared to the *Multicommodity flow* model described in the previous section, that suffers from computational limitations because of the large number of variables and constraints (see section 3.6.1.2). The models proposed in this section are based on the U^t model (see section 3.4.2.2) proposed for the PCMCA, but replacing the connectivity constraints (3.17) with a modified version of the subtour elimination constraints for the *Travel-*

ing Salesman Problem proposed by Miller, Tucker, and Zemlin in [67]. The rest of this section is organized as follows. Section 4.4.2.1 introduces a *complete model* for the PCMCA-WT that extends the U^t model proposed for the PCMCA, but uses a polynomial set of constraints to eliminate subtours and ensure the connectivity of the solution. Section 4.4.2.2 introduces a *reduced model* for the PCMCA-WT that uses a smaller number of variables and constraints compared to the *complete model*, by exploiting a special property of the PCMCA-WT.

4.4.2.1 Complete Model

Let x_{ij} be a variable associated with every arc $(i,j) \in A$ such that $xij = 1$ if $(i,j) \in T$, and 0 otherwise. Let y_i be a variable associated with every vertex $i \in V$ that indicates the order in which vertex i is visited on the path connecting vertex i to the root r. Let u_j^t be a variable associated with every vertex $j \in V$, and vertex $t \in V$ where t is part of a precedence relationship (i.e. $\exists(s,t) \in R$). Let d_j be the time at which the flow enters vertex $j \in V$, and let w_j be the waiting time before the flow enters vertex j. The PCMCA-WT can be modeled as the following MILP model.

$$\text{minimize} \sum_{(i,j)\in A} c_{ij}x_{ij} + \sum_{i\in V} w_i \tag{4.20}$$

$$\text{subject to:} \sum_{(r,j)\in A} x_{rj} \geq 1 \tag{4.21}$$

$$\sum_{(i,j)\in A} x_{ij} = 1 \qquad \forall j \in V\backslash\{r\} \tag{4.22}$$

$$y_i - y_j + 1 \leq n(1 - x_{ij}) \qquad \forall (i,j) \in A : j \neq r \tag{4.23}$$

$$u_s^t = 0 \qquad \forall (s,t) \in R \tag{4.24}$$

$$u_t^t = 1 \qquad \forall t \in V : \exists (s,t) \in R \tag{4.25}$$

$$u_j^t - u_i^t - x_{ij} \geq -1 \qquad \forall t \in V : \exists (s,t) \in R, (i,j) \in A \qquad (4.26)$$

$$d_r = 0 \qquad (4.27)$$

$$w_r = 0 \qquad (4.28)$$

$$d_j \geq d_i - M + (M + c_{ij})x_{ij} \qquad \forall (i,j) \in A \qquad (4.29)$$

$$w_j \geq d_j - d_i - M + (M - c_{ij})x_{ij} \qquad \forall (i,j) \in A \qquad (4.30)$$

$$d_t \geq d_s \qquad \forall (s,t) \in R \qquad (4.31)$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A \qquad (4.32)$$

$$y_i \geq 0 \qquad \forall i \in V \qquad (4.33)$$

$$u_j^t \geq 0 \qquad \forall t \in V : \exists (s,t) \in R, j \in V \qquad (4.34)$$

$$d_i, w_i \in \mathbb{R}^+_{\leq M} \qquad \forall i \in V \qquad (4.35)$$

Constraint (4.21) enforces that there must be at least one arc leaving the root that enters a vertex other than the root. The set of constraints (4.22) enforces that every vertex other than the root must have exactly one incoming arc. Constraints (4.23) are the subtour elimination constraints, which enforce that any feasible solution must not contain any cycles. The set of constraints (4.21)-(4.23) work together in order to guarantee that any feasible solution is an arborescence rooted at vertex r. Constraints (4.24) and (4.25) fix the values of u_s^t and u_t^t to 0 and 1 respectively, for all $(s,t) \in R$, and $t \in V : \exists (s,t) \in R$. Constraints (4.26) impose that $u_j^t \geq u_i^t$ if $x_{ij} = 1$. Constraint (4.27) sets the distance from the root r to itself to be equal to 0. Constraint (4.28) sets the waiting time at the root r to be equal to 0. Constraints (4.29) impose that when arc $(i,j) \in A$ is selected to be part of the arborescence, then the time at which the flow enters vertex j is greater than or equal to the time at which the flow enters vertex i plus c_{ij}. Constraints (4.30) enforce that the waiting time at vertex j is greater than or equal to the difference between the time at which

the flow enters vertex j and the time at which the flow enters vertex i plus c_{ij}. Constraints (4.31) enforce that the time at which the flow enters vertex t is greater than or equal to the time at which the flow enters vertex s for all $(s,t) \in R$. Finally, constraints (4.32)-(4.35) define the domain of the variables.

4.4.2.2 Reduced Model

Removing the set of constraints (4.24)-(4.26) from the model described in section 4.4.2.1, enforces the precedence relationships between s and t with $(s,t) \in R$, except for a special case. A directed path which visits t before visiting s, implies that $d_t \geq d_s$, which violates the set of constraints (4.31). However, the set of constraints (4.31) might fail to enforce a precedence relationship between s and t, if there exists a zero-cost (t,s)-path in G. Therefore, we only need to define the set of constraints (4.24)-(4.26) for all t, where there exists a zero-cost (t,s)-path in G.

A zero-cost (t,s)-path in G for some $(s,t) \in R$ can be found using a *Depth-First-Search* (DFS) algorithm [83] with *backtracking* [85] as described in Algorithm 7, and it is explained as follows. Line 2 creates an empty set K which will eventually contain every vertex $t \in V$ such that t is part of a precedence relationship, and s can be reached from t through a zero-cost path for all $(s,t) \in R$. Line 3 creates an array of size $|V|$ and marks every vertex $v \in V$ as not visisted. The purpose of this array is to avoid visiting a vertex more than once on the same path if the graph G contain cycles. Lines 4-7 calls the *Check_Path* procedure for all $(s,t) \in R$ and initializes the current cost of the path to zero. Line 8 returns the set K. Line 12 sets the state of the current vertex u as visited. Lines 13-17 add vertex t to the set K if the current vertex u is the target vertex s, and the cost of the current path is equal to zero. Lines 18-22 call the *Check_Path* procedure for the neighboring vertices of the current vertex u that are not visited, and if the cost of the current path is equal to zero. The condition at line 19 avoids visiting a neighboring vertex whenever the

Algorithm 7 An Algorithm that Finds all t that are Part of a Zero-Cost (t, s)-path

```
1: procedure FIND_t(R)
2:     K = φ
3:     visited[v] = false ∀v ∈ V
4:     for all (s, t) ∈ R do
5:         cost = 0
6:         Check_Path(G, t, t, s, cost)
7:     end for
8:     return K
9: end procedure
10:
11: procedure CHECK_PATH(G, u, t, s, cost)
12:     visited[u] = true
13:     if u = s AND cost = 0 then
14:         visited[u] = false
15:         K = K ∪ t
16:         return
17:     end if
18:     for all (u, v) ∈ A do
19:         if cost = 0 AND visited[v] = false then
20:             Check_Path(G, v, t, s, cost + c_uv)
21:         end if
22:     end for
23:     visited[t] = false
24: end procedure
```

cost of the current path is greater than zero, which means that it is impossible for t to be part of a zero-cost path that reaches s as the current cost of the path is greater than zero. Algorithm 7 has a computational complexity of $O(|V|!)$. However, the instances for the problem analyzed normally contain a very small number of zero weight arcs due to their nature, and the procedure tend to terminate in the first few iterations for the majority of $t \in V$. This is true for all the instances considered in the experiments reported in section 4.6.

Compared to the model introduced in section 4.4.2.1 the *Reduced* model uses

a smaller number of variables and constraints, thus reducing its memory footprint. This model is much easier to solve in theory, but has the weakness of potentially providing a weaker linear relaxation, on top of having the drawback of a preprocessing phase.

Let $P_{ij} \subset A$ be a simple directed (i,j)-path, and let $c(P_{ij}) = \sum_{(i,j)\in P} c_{ij}$ be the cost of that path. Let $V_s = \{t \in V \backslash \{r\} \,|\, \exists (s,t) \in R, c(P_{ts}) = 0\}$. Applying the aforementioned reduction would results in the following MILP model for the PCMCA-WT.

$$\text{minimize} \sum_{(i,j)\in A} c_{ij}x_{ij} + \sum_{i\in V} w_i \tag{4.36}$$

$$\text{subject to:} \sum_{(r,j)\in A} x_{rj} \geq 1 \tag{4.37}$$

$$\sum_{(i,j)\in A} x_{ij} = 1 \qquad \forall j \in V\backslash\{r\} \tag{4.38}$$

$$y_i - y_j + 1 \leq n(1 - x_{ij}) \qquad \forall (i,j) \in A : j \neq r \tag{4.39}$$

$$u_s^t = 0 \qquad \forall (s,t) \in R : t \in V_s \tag{4.40}$$

$$u_t^t = 1 \qquad \forall t \in V_s \tag{4.41}$$

$$u_j^t - u_i^t - x_{ij} \geq -1 \qquad \forall (s,t) \in R : t \in V_s, (i,j) \in A \tag{4.42}$$

$$d_r = 0 \tag{4.43}$$

$$w_r = 0 \tag{4.44}$$

$$d_j \geq d_i - M + (M + c_{ij})x_{ij} \qquad \forall (i,j) \in A \tag{4.45}$$

$$w_j \geq d_j - d_i - M + (M - c_{ij})x_{ij} \qquad \forall (i,j) \in A \tag{4.46}$$

$$d_t \geq d_s \qquad \forall (s,t) \in R \tag{4.47}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A \tag{4.48}$$

$$y_i \geq 0 \qquad \forall i \in V \tag{4.49}$$

$$u_j^t \geq 0 \qquad \forall t \in V_s, j \in V \quad (4.50)$$

$$d_i, w_i \in \mathbb{R}^+_{\leq M} \qquad \forall i \in V \quad (4.51)$$

4.4.3 Distance-Accumulation Model

In this section we describe a MILP model for the PCMCA-WT that extends the Set-Based model for the PCMCA problem previously described in section 3.4.4. The model is called distance-accumulation model as it computes the distance of each vertex from the root by accumulating the value over the path.

$$\text{minimize} \sum_{(i,j)\in A} c_{ij}x_{ij} + \sum_{j\in V} w_j \quad (4.52)$$

$$\text{subject to} \sum_{(i,j)\in A} x_{ij} = 1 \qquad \forall j \in V\backslash\{r\} \quad (4.53)$$

$$\sum_{(i,k)\in\delta^-(S)} x_{ik} \geq 1 \qquad \forall j \in V\backslash\{r\}, \forall S \subseteq V_j\backslash\{r\} : j \in S \quad (4.54)$$

$$d_r = 0 \quad (4.55)$$

$$w_r = 0 \quad (4.56)$$

$$d_j \geq d_i - M + (M + c_{ij})x_{ij} \qquad \forall (i,j) \in A \quad (4.57)$$

$$w_j \geq d_j - d_i - M + (M - c_{ij})x_{ij} \qquad \forall (i,j) \in A \quad (4.58)$$

$$d_t \geq d_s \qquad \forall (s,t) \in R \quad (4.59)$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A \quad (4.60)$$

$$d_i, w_i \geq 0 \qquad \forall i \in V \quad (4.61)$$

Constraints (4.53) impose the first property of an arborescence, namely that every vertex $v \in V\backslash\{r\}$ must have a single parent. Constraints (4.54) model the connectivity constraint, that is every vertex $v \in V\backslash\{r\}$ must be reachable from

the root, and they also impose the precedence constraints where the resulting arborescence should not include a (t, s)-path for all $(s, t) \in R$. This will lead to an arborescence such that the flow never enters t before entering s, if s and t belong to the same directed path. Constraint (4.55) sets the distance from the root r to itself to be equal to 0. Constraint (4.56) set the waiting time at the root vertex to 0. Constraints (4.57) impose that when arc $(i, j) \in A$ is selected to be part of the arborescence, then the time at which the flow enters vertex j is greater than or equal to the time at which the flow enters vertex i plus c_{ij}. Constraints (4.58) enforce that the waiting time at vertex j is greater than or equal to the difference between the time at which the flow enters vertex j and the time at which the flow enters vertex i plus c_{ij}. Constraints (4.59) enforce that the time at which the flow enters vertex t is greater than or equal to the time at which the flow enters vertex s for all $(s, t) \in R$. Finally, constraints (4.60) and (4.61) define the domain of the variables. Constraints (4.54) are dynamically added to the model using the same separation procedure described by algorithm 4 in section 3.4.4.

4.4.4 Adjusted Arc-Cost Model

In this section we propose a MILP model that originates from removing inequalities (4.58) from the model introduced in section 4.4.3 and representing the value of w_j by the nonlinear term

$$w_j = \sum_{i:(i,j)\in A} (d_j - d_i - c_{ij})x_{ij} \tag{4.62}$$

A different MILP model is then derived as follows.

Proposition 1. *The waiting time at vertex $j \in V$ can be expressed by the nonlinear*

equality (4.62).

Proof. Inequalities (4.58) can be rewritten as $w_j \geq d_j - d_i - c_{ij} - M(1 - x_{ij})\, \forall (i,j) \in A$. If $x_{ij} = 0$ then w_j has to be greater than or equal to a negative value, however the value of w_j should be greater than or equal to zero by definition. Accordingly, the inequality would be active and affect the solution only when $x_{ij} = 1$. Therefore, we can represent the waiting time at vertex j using equality (4.62). □

Based on Proposition 1, we can replace the second term in the objective function (4.52) as follows:

$$\sum_{j \in V} w_j = \sum_{j \in V} \sum_{i:(i,j) \in A} (d_j - d_i - c_{ij}) x_{ij} = \sum_{(i,j) \in A} (d_j - d_i - c_{ij}) x_{ij}$$

This means that inequalities (4.58) are no longer necessary as the objective function no longer depends on w, which results in the following nonlinear model.

$$\text{minimize} \sum_{(i,j) \in A} c_{ij} x_{ij} + \sum_{(i,j) \in A} (d_j - d_i - c_{ij}) x_{ij} \tag{4.63}$$

$$\text{subject to} \sum_{(i,j) \in A} x_{ij} = 1 \qquad \forall j \in V \backslash \{r\} \tag{4.64}$$

$$\sum_{\substack{(i,k) \in A: \\ i \in V_j \backslash S,\, k \in S}} x_{ik} \geq 1 \qquad \forall j \in V \setminus \{r\}, \forall S \subseteq V_j \backslash \{r\} : j \in S \tag{4.65}$$

$$d_r = 0 \tag{4.66}$$

$$d_j \geq d_i - M + (M + c_{ij}) x_{ij} \qquad \forall (i,j) \in A \tag{4.67}$$

$$d_t \geq d_s \qquad \forall (s,t) \in R \tag{4.68}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A \tag{4.69}$$

$$d_i \geq 0 \qquad \forall i \in V \tag{4.70}$$

Proposition 2. *Using a new set of $|A|$ variables z and $2|A|$ new constraints, the objective function (4.63) can be linearized as follows:*

$$minimize \sum_{j \in V \setminus \{r\}} d_j - \sum_{(i,j) \in A} z_{ij}$$

Proof. The objective function (4.63) can be rewritten as follows:

$$\begin{aligned} &\sum_{(i,j) \in A} c_{ij} x_{ij} + \sum_{(i,j) \in A} (d_j - d_i - c_{ij}) x_{ij} = \\ &\sum_{(i,j) \in A} d_j x_{ij} - \sum_{(i,j) \in A} d_i x_{ij} \; = \sum_{j \in V \setminus \{r\}} d_j - \sum_{(i,j) \in A} d_i x_{ij} \end{aligned} \tag{4.71}$$

We use the fact that $\sum_{(i,j) \in A} d_j x_{ij} = \sum_{j \in V \setminus \{r\}} d_j$ as each $j \in V \setminus \{r\}$ has exactly one x_{ij} assigned to 1 in an arborescence, as imposed by (4.53).
Since the term $d_i x_{ij}$ is summed over each arc $(i, j) \in A$, then we need at least $2|A|$ constraints to linearize the product. We can substitute each term $d_i x_{ij}$ by a new continuous variable z_{ij} and the following two inequalities:

$$z_{ij} \leq M x_{ij} \qquad \forall (i,j) \in A \tag{4.72}$$

$$z_{ij} \leq d_i \qquad \forall (i,j) \in A \tag{4.73}$$

Inequalities (4.72) ensure that if $x_{ij} = 0$ then $z_{ij} = 0$. On the other hand, if $x_{ij} = 1$, then inequalities (4.72) ensure that z_{ij} is less than the upper bound on the optimal solution which is further tightened by inequalities (4.73). This results in a total of $2|A|$ new constraints and (4.63) can now be expressed as $\sum_{j \in V \setminus \{r\}} d_j - \sum_{(i,j) \in A} z_{ij}$ by elaborating on (4.71). □

Based on Proposition 2, we can derive the following MILP model.

$$\text{minimize} \sum_{j \in V \setminus \{r\}} d_j - \sum_{(i,j) \in A} z_{ij} \tag{4.74}$$

$$\text{subject to} \sum_{(i,j) \in A} x_{ij} = 1 \qquad \forall j \in V \setminus \{r\} \tag{4.75}$$

$$\sum_{\substack{(i,k) \in A: \\ i \in V_j \setminus S,\, k \in S}} x_{ik} \geq 1 \qquad \forall j \in V \setminus \{r\}, \forall S \subseteq V_j \setminus \{r\} : j \in S \tag{4.76}$$

$$d_r = 0 \tag{4.77}$$

$$d_j \geq d_i - M + (M + c_{ij}) x_{ij} \qquad \forall (i,j) \in A \tag{4.78}$$

$$d_t \geq d_s \qquad \forall (s,t) \in R \tag{4.79}$$

$$z_{ij} \leq d_i \qquad \forall (i,j) \in A \tag{4.80}$$

$$z_{ij} \leq M x_{ij} \qquad \forall (i,j) \in A \tag{4.81}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A \tag{4.82}$$

$$z_{ij} \geq 0 \qquad \forall (i,j) \in A \tag{4.83}$$

$$d_i \geq 0 \qquad \forall i \in V \tag{4.84}$$

Proposition 3. *The following inequalities are valid for the Adjusted Arc-Cost model:*

$$\sum_{i \in V : (i,j) \in A} z_{ij} \leq d_j - \sum_{(i,j) \in A} c_{ij} x_{ij} \qquad \forall j \in V \setminus \{r\} \tag{4.85}$$

Proof. Since for each vertex $j \in V \setminus \{r\}$ there is only one active arc $(i,j) \in A$ entering j (from inequalities (4.75)), from inequalities (4.78) we can derive the following new

quadratic inequalities:

$$d_j \geq \sum_{i \in V:(i,j) \in A} d_i x_{ij} + \sum_{(i,j) \in A} c_{ij} x_{ij} \qquad \forall j \in V \backslash \{r\} \tag{4.86}$$

From inequalities (4.80) and (4.81) we have $z_{ij} \leq d_i x_{ij}$ (see Proposition 2), then inequality (4.85) can be derived from inequality (4.86) as follows.

$$\begin{aligned} d_j \geq \sum_{i \in V:(i,j) \in A} d_i x_{ij} + \sum_{(i,j) \in A} c_{ij} x_{ij} &\implies \sum_{i \in V:(i,j) \in A} d_i x_{ij} \leq d_j - \sum_{(i,j) \in A} c_{ij} x_{ij} \\ &\implies \sum_{i \in V:(i,j) \in A} z_{ij} \leq d_j - \sum_{(i,j) \in A} c_{ij} x_{ij} \\ &\implies d_j \geq \sum_{i \in V:(i,j) \in A} z_{ij} + \sum_{(i,j) \in A} c_{ij} x_{ij} \end{aligned}$$

□

It should be noted that inequalities (4.85) are not an integral part of the *Adjusted Arc-Cost* model, but are added to have a stronger linear relaxation. If the inequalities are not included in the model, then the value of the z_{ij}s can be substantially larger than the value of the d_js in order to minimize the value of the objective function. This could result in feasible solutions of the linear relaxation with a negative objective function. This would make the MILP much harder to solve. Therefore, inequalities (4.85) are considered for all the experiments reported section 4.6.

4.5 Constraint Programming Models

In this section we introduce two CP models for the PCMCA-WT that are based on the same intuitions behind the *Complete* and *Reduced Path-Based* MILP models described previously in sections 4.4.2.1 and 4.4.2.2. The two *Path-Based* MILP models are reformulated as CP models for the following reasons. Unlike MILP

solvers, the CP solver used in this work does not allow us to dynamically add constraints to the model once they are violated. This means that using a model with an exponential set of constraints requires a substantial amount of preprocessing time. Furthermore, the *Path-Based* MILP models perform substantially better on the benchmark instances compared to the the *Multicommodity flow model* introduced in section 4.4.1 (see section 4.6).

The two *Path-Based* MILP models use a *big-M* formulation in order to describe the nonlinear relation between the variable x_{ij} and the set of variables $\{y_i, u_j^t, d_j, w_j\}$ in order to turn the constraints off whenever the value of x_{ij} is equal to zero. Such formulations add an extra layer of complexity, which is finding a feasible value for M that is large enough to guarantee the correctness of the optimal solution. Moreover, M is part of the coefficient matrix, and mixing very large coefficients (i.e. M) with much smaller coefficients can create numerical instability, leading the solver to spend more time computing linear program pivots.

A feature that is available in some MILP solvers and CP solvers, is that they support logical constraints, that allows us to devise a model for the PCMCA-WT which does not use a *big-M* formulation. However, unlike MILP solvers which uses a combination of relaxations (strengthened by cutting-planes) and branch-and-bound, a CP solver makes decisions on variables and values, and after each decision, performs a set of logical inferences to reduce the available options for the remaining variables domains. For many problem classes, logical inferencing is less powerful than solving a continuous relaxation of the model, since the solution of a continuous relaxation can guide the search and provide dual bounds. However, the propagation schemes can be effective when the relaxation is extremely time consuming to solve or when the combinatorial structure of the model gets lost in the relaxation [46]. Preliminary results have clearly shown that the CP solver takes advantage of logical constraints, while this does not happen for the MILP solver, especially on large

sized instances where the value of *big-M* is considerably larger than the value of the optimal solution.

4.5.1 Complete Model

Using a set of logical constraints, and implication constraints which force the implied constraint if the value of the variable is true, the *Complete Path-Based* MILP model introduced in section 4.4.2.1 for the PCMCA-WT can be modeled as the following CP model.

$$\text{minimize} \sum_{(i,j)\in A} c_{ij}x_{ij} + \sum_{i\in V} w_i \tag{4.87}$$

$$\text{subject to:} \sum_{(i,j)\in A} x_{ij} = 1 \qquad \forall j \in V\backslash\{r\} \tag{4.88}$$

$$x_{ij} \implies y_j = y_i + 1 \qquad \forall (i,j) \in A : j \neq r \tag{4.89}$$

$$u_s^t = 0 \qquad \forall (s,t) \in R \tag{4.90}$$

$$u_t^t = 1 \qquad \forall t \in V : \exists (s,t) \in R \tag{4.91}$$

$$x_{ij} \implies u_j^t \geq u_i^t \qquad \forall t \in V : \exists (s,t) \in R, (i,j) \in A \tag{4.92}$$

$$d_r = 0 \tag{4.93}$$

$$w_r = 0 \tag{4.94}$$

$$x_{ij} \implies d_j = d_i + w_j + c_{ij} \qquad \forall (i,j) \in A \tag{4.95}$$

$$d_t \geq d_s \qquad \forall (s,t) \in R \tag{4.96}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A \tag{4.97}$$

$$y_i \geq 0 \qquad \forall i \in V \tag{4.98}$$

$$u_j^t \geq 0 \qquad \forall t \in V : \exists (s,t) \in R, j \in V \tag{4.99}$$

$$d_i, w_i \in \mathbb{Z}^+_{\leq M} \qquad \forall i \in V \tag{4.100}$$

Constraints (4.88) enforce that every vertex other than the root must have exactly one arc entering it. Constraints (4.88) plus (4.89) are the subtour elimination constraints that model the nonlinear relationship $y_j = (y_i + 1)x_{ij}$, by setting the value of y_j to $y_i + 1$ iff $x_{ij} = 1$ (true). Constraints (4.92) are the precedence enforcing constraints, that set the value of u_j^t to be greater than or equal to u_j^t iff $x_{ij} = 1$, and model the nonlinear relationship $u_j^t \geq u_i^t x_{ij}$. Constraints (4.95) set the value of d_j, to $d_i + w_j + c_{ij}$ iff $x_{ij} = 1$. The set of constraints (4.95) model the nonlinear relationship $(d_j - d_i - w_j - c_{ij})x_{ij} = 0$. Note that by using a *channeling constraints* [46] we are able to combine the two constraints (4.29) and (4.30) into a single equality which computes the value of d_j. Constraints (4.96) enforce that the flow enters vertex t at a time step which is greater than or equal to the time step at which the flow enters vertex s for all $(s, t) \in R$. Finally, constraints (4.97)-(4.100) define the domain of the variables.

Note that variables d_i and w_i are defined as integers (compared to the MILP model), since a CP solver only accepts integer variables and coefficients. This means that c_{ij} for all $(i, j) \in A$ should be integer or to be discretized before solving the model. The value of c_{ij} can be discretized by multiplying every c_{ij} by a constant k, and then considering only the integer part of the result. In order to compute the correct solution cost, the objective function value should be divided by k. A higher k value leads to higher numerical precision, whereas a low k value leads to a lower numerical precision and thus faster execution. Therefore, a k value which balances the two factors should be considered.

The difference between the *Complete Path-Based* MILP model introduced in Section 4.4.2.1 and the CP model introduced here, is the set of constraints (4.89), (4.92), and (4.95) that eliminate the use of *big-M*, and combine the two constraints (4.29) and (4.30) into one single equality constraint.

4.5.2 Reduced Model

Using the same reasoning explained in section 4.4.2.2, the size of the set of u_j^t variables and the size of the set of constraints (4.92) can be reduced, resulting in the following *Reduced* CP model.

$$\text{minimize} \sum_{(i,j)\in A} c_{ij}x_{ij} + \sum_{i\in V} w_i \tag{4.101}$$

$$\text{subject to:} \sum_{(i,j)\in A} x_{ij} = 1 \qquad \forall j \in V\backslash\{r\} \tag{4.102}$$

$$x_{ij} \implies y_j = y_i + 1 \qquad \forall (i,j) \in A : j \neq r \tag{4.103}$$

$$u_s^t = 0 \qquad \forall (s,t) \in R : t \in V_s \tag{4.104}$$

$$u_t^t = 1 \qquad \forall t \in V_s \tag{4.105}$$

$$x_{ij} \implies u_j^t \geq u_i^t \qquad \forall (s,t) \in R : t \in V_s, j \in V\backslash\{r\} \tag{4.106}$$

$$d_r = 0 \tag{4.107}$$

$$w_r = 0 \tag{4.108}$$

$$x_{ij} \implies d_j = d_i + w_j + c_{ij} \qquad \forall (i,j) \in A \tag{4.109}$$

$$d_t \geq d_s \qquad \forall (s,t) \in R \tag{4.110}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A \tag{4.111}$$

$$y_i \geq 0 \qquad \forall i \in V \tag{4.112}$$

$$u_j^t \geq 0 \qquad \forall t \in V_s, j \in V \tag{4.113}$$

$$d_i, w_i \in \mathbb{Z}_{\leq M}^{+} \qquad \forall i \in V \tag{4.114}$$

4.6 Computational Results

In this section we present the experiments conducted to evaluate the several models proposed in sections 4.4 and 4.5. In section 4.6.1 we evaluate and discuss the performance of the MILP models introduced in section 4.4. In section 4.6.2 we evaluate and discuss the performance of the CP models introduced in section 4.5, and compare them to the performance of the *Path-Based models* introduced in section 4.4.2.

The computational experiments are based on the benchmark instances of TSPLIB [75], SOPLIB [69], and COMPILERS [79] originally proposed for the Sequential Ordering Problem [27], and are modified as explained in section 3.6. The benchmark sets contain a total of 116 instances, ranging in size between 9 and 700 vertices, with an average of 248 vertices.

4.6.1 The MILP Models

All the experiments in this section are performed on a laptop with an Intel i7-8550U processor running at 1.8 GHz with 8 GB of RAM. The MILPs are solved using CPLEX 12.8 [53]. CPLEX is run with the two parameters *NodeSelect* and *MIP emphasis* set to *BestBound* and *MIPEmphasisOptimality* respectively, and single threaded standard Branch-and-Cut (B&C) algorithm is applied for solving the MILP models. A time limit of one hour is set on the computation time of each computational method/instance. The models have been implemented in C++ 11, and are compiled with Microsoft C/C++ Optimizing Compiler v19.

For the rest of this section we will be referring to the *Mutlicommodity Flow model* from section 4.4.1 as (*MCF*), the *Complete Path-Based model* from section 4.4.2.1 as (*CPB*), the *Reduced Path-Based model* from section 4.4.2.1 as (*RPB*), the *Distance-Accumulation model* from section 4.4.3 as (*DA*), and the *Adjusted Arc-Cost model*

Table 4.1: PCMCA-WT summary of the results of solving the models *CPB, RPB, DA* and *AAC* for SOPLIB instances.

	Model			
	CPB	RPB	DA	AAC
Average LR optimality gap	62.9%	63.2%	62.6%	63.0%
Average IP optimality gap	78.2%	68.5%	68.1%	67.7%
Average LR solution time	223.6	184.6	161.4	677.3
Average IP solution time	-	132.5	287.2	45.7
Best LR solution	22	15	27	16
Best lower bound	5	32	1	1
Best upper bound	2	17	10	10
Optimal solution	4	8	10	9

from section 4.4.4 as (AAC).

In this section, we omit the results of the PCMCA-WT for SOPLIB instances from the discussion. However, we can draw the following conclusions on these instances. The model *MCF* is unable to solve large sized instances due to memory issues (building the model consumes around 5GB of memory on average) or time out while solving the model's linear relaxation. Since the linear relaxation of the model *MCF* is unable to solve a single instance from SOPLIB benchmark set, we concluded that it is highly unsuitable for solving such instances, and therefore the results are omitted from the discussion. The results for the rest of the models are summarized in table 4.1, where we report the following. The *Average LR optimality gap* reports the average optimality gap achieved by solving the linear relaxation of the respective model, and is computed as $100 \cdot \frac{Cost_{Best} - Cost_{LR}}{Cost_{Best}}$, where $Cost_{Best}$ is the best known upper bound for the instances, and $Cost_{LR}$ is the value of the objective function of the respective linear relaxation model. The *Average IP optimality gap* reports the average optimality gap achieved by solving the integer program (IP) of the respective model. The *Average LR solution time* reports the average solution

time in seconds of solving the linear relaxation of the respective model, for a total of 68 instances. The *Average IP solution time* reports the average solution time in seconds for the instances that are optimally solved by the three models *RPB, DA*, and *AAC* (6 instances out of a total of 48 instances). The *Average IP solution time* for the model *CPB* is omitted, as it optimally solves only 4 of those instances and is 152.7% slower on average compared to the model *RPB*. The *Best LR solution* reports the number of instances where solving the linear relaxation of the respective model results in the highest solution cost (best lower bound). The *Best lower bound* reports the number of instances where the IP model was able to find the best lower bound estimate on the value of the optimal solution. The *Best upper bound* reports the number of instances where the IP model was able to find the best new solution cost, and does not include the instances where an optimal solution is found. Finally, *Optimal solution* reports the number of instances that were solved optimaly by the IP model.

Considering the linear relaxation of the models, all five models achieve a similar average optimality gap, however the model *DA* has the lowest average solution time and finds the best solution cost for 56% of the instances. This shows that on average, the model *DA* has the strongest linear relaxation out of the five models considered. On the other hand, when considering the IP models, the model *AAC* is 84.1% faster on average, and achieves better solutions for 45.8% of the instances compared to the model *DA*. More specifically, the model *AAC* performs better than the model *DA* at solving large sized instances with dense precedence graphs. However, the model *RPB* has the best overall performance on SOPLIB instances, as it has an average solution time that falls in between the two models *DA* and *AAC*, and is able to find the best lower and upper bounds for a larger number of instances, with a slightly higher average optimality gap. The aforementioned results are generally worse compared to the other two benchmark sets as will be shown later, which indicates that large-

Table 4.2: PCMCA-WT summary of the results of solving the linear relaxation of the models *MCF, CPB, RPB, DA* and AAC for TSPLIB and COMPILERS instances.

	Model				
	MCF	CPB	RPB	DA	AAC
Average optimality gap	19.4%	23.2%	23.2%	18.3%	18.5%
Average solution time	889.9	4.6	3.6	19.6	38.4
Average number of cuts	-	11	-	97	105
Best LR solution	5	22	20	20	6

sized instances with a highly dense precedence graph are outside the reach of the models proposed due to the intrinsic complexity of the problem.

4.6.1.1 The Linear Relaxation of the Models

Table 4.2 summarizes the results of solving the models' linear relaxation considering 58 out of a total of 68 instances that are solved by all five models. The *Average optimality gap* reports the average optimality gap achieved by solving the linear relaxation of the respective model. The *Average solution time* reports the average solution time in seconds spent at solving the respective model's linear relaxation. The *Average number of cuts* reports the average number of model specific inequalities that are dynamically added to the model. The *Best LR solution* reports the number of instances where solving the respective model's linear relaxation results in the highest lower bound estimate.

In summary, the linear relaxation of the model *MCF* is unable to solve instances larger than 240 vertices. Compared to the model *MCF*, the model *CPB* results in a 19.6% increase in the average optimality gap, the model *RPB* results in a 19.1% increase in the average optimality gap, the model *DA* results in a 5.7% decrease in the average optimality gap, and the model *AAC* results in a 4.6% decrease in the average optimality. Furthermore, compared to the model *MCF*, the model *CPB* is

99.5% faster on average, the model *RPB* is 99.6% faster on average, the model *DA* is 97.8% faster on average, and the model *AAC* is 95.7% faster on average. Considering all factors, the results show that the model *DA* has the strongest linear relaxation out of the five models considered. However, when excluding the model *MCF*, and considering 67 out of 68 instances that are solved by all four models, the same conclusion holds, which is that the model *DA* has the strongest linear relaxation, and achieves the smallest average optimality gap of 22.1% with an average solution time of 19.6 seconds. In terms of the number of cuts that are dynamically added to the model, the model *DA* generates 781.8% more cuts compared to the model *CPB*, while the model *AAC* generates 854.5% more cuts compared to the model *CPB*.

A major problem that we can notice in the model *MCF* is that the solution times are considerably larger when compared to the other four models. For example, the model *MCF* finds the optimal solution of the instance ESC78 within 9 minutes compared to 6.7, 4.8, 4.5 and 6.0 seconds of computing time by the models *CPB*, *RPB*, *DA*, and *AAC* respectively. The same increased solution time can be noticed in other instances, sometimes reaching almost an hour to solve the linear relaxation compared to few seconds. Comparing the two models *DA* and *AAC*, the *AAC* model finds better estimates on the symmetrical COMPILERS instances which have symmetric costs.

In general, it is hard to decide which linear relaxation would perform better on some instances, however the model *DA* seems to be the most suitable, as its linear relaxation is stronger on average the other models.

4.6.1.2 The IP Models

Table 4.3 summarizes the results of solving the IP models, where we report the following. The *Average optimality gap* reports the average optimality gap achieved by solving the respective model when considering 58 out of a total of 68 instances

Table 4.3: PCMCA-WT summary of the results of solving the MILP models *MCF, CPB, RPB, DA* and AAC for TSPLIB and COMPILERS instances.

	Model				
	MCF	CPB	RPB	DA	AAC
Average optimality gap	29.4%	19.4%	15.5%	13.8%	15.7%
Average solution time	483.3	561.9	55.8	213.7	98.0
Average number of cuts	-	0	-	434	1680
Best lower bound	1	5	22	21	6
Best upper bound	2	8	22	17	15
Optimal solution	11	13	16	17	16

that are solved by all five models within the time limit. The *Average solution time* reports the average solution time in seconds of the instances that are optimally solved by all five models (a total of 11 out of 68). The *Average number of cuts* reports the average number of model specific inequalities that are dynamically added to the model for the instances that are optimally solved by all five models (a total of 12 out of 68). The *Best lower bound* reports the number of instances where the respective model was able to find the best lower bound estimate on the value of the optimal solution. The *Best upper bound* reports the number of instances where the respective model was able to find the best solution cost. Finally, *Optimal solution* reports the number of instances that were solved optimaly by the respective model.

Compared to the model *MCF*, the model *CPB* results in a 34.0% improvement in the average optimality gap, the model *RPB* results in a 47.3% improvement in the average optimality gap, the model *DA* results in a 53.1% improvement in the average optimality gap, and the model *AAC* results in a 46.6% improvement in the average optimality. In terms of average solution time, and compared to the model *MCF*, the model *CPB* is 16.3% slower on average, the model *RPB* is 88.5% faster on average, the model *DA* is 55.8% faster on average, and the model *AAC* is

79.7% faster on average. Furthermore, The model AAC generates 287.1% more cuts compared to the model DA, while the model CPB does not dynamically adds any cuts on average to the model, which shows that the precedence-enforcing constraints are rarely violated for in the model.

Finally, we compare the average number of nodes that are generated in the search-decision tree for a total of 9 instances that are optimally solved by all five models. Compared to the model MCF, the model CPB generates 3034.94% more nodes, the model RPB generates 341.1% more nodes, the model DA generates 96.7% more nodes, and the model AAC generates 406.3% more nodes. We can conclude that the two models RPB and DA overall perform the best out of the five proposed models.

Based on the computational results, the following conclusions can be drawn. The model DA performs better compared to the model AAC, except on symmetrical instances and/or instances with extreme high densities larger than 0.9. Out of a total of 68 instances, and considering the overall best two models (RPB and DA) in terms of achieved average optimality gap and solution time, the model RPB was able to find the best lower bound for 22 instances, the best upper bound for 22 instances, and the optimal solution for 16 instances. On the other hand, the model DA was able to find the the best lower bound for 21 instances, the best upper bound for 17 instances, and the optimal solution for 17 instances. This shows that on average, the model RPB performs better than the model DA. However, the model DA often performs better on instances with medium density precedence graphs.

4.6.1.3 Overall Results

In this section we report the overall results of the models discussed in section 4.6.1. The overall results can be found in tables 4.4-4.9 where we report the following. Columns *Name* and *Size* report the name and size of the instance, $\rho(R)$ reports the

density of arcs in the set of precedence relationships R, and is computed as $\frac{2 \cdot |R|}{|V|(|V|-1)}$. For each model we report the following columns. Column *Cost* reports the value of the objective function of the linear relaxation of the model. Columns *LB* and *UB* report the lower and upper bound on the value of the optimal solution obtained from solving the respective MILP model. Column *Cuts* reports the number of model-dependent cuts (inequalities) that are dynamically added to the model. Column *Nodes* reports the number of nodes generated in the search decision-tree. Column *Gap* indicates the percentage relative difference between the cost of the best-known integer solution of the instance ($Cost_{Best}$), and the objective function cost of the model's linear relaxation ($Cost$), computed as $100 \cdot \frac{Cost_{Best} - Cost}{Cost_{Best}}$. Column *IP Gap* measures the percentage relative difference between the upper and lower bound obtained from solving the respective model, calculated as $100 \cdot \frac{UB - LB}{UB}$. Column *Time [s]* reports the solution time in seconds. The solution time is not reported in the tables for the instances that are not optimally solved within the time limit. Furthermore, the solution information is not reported for instances where it was not possible to solve the model's linear relaxation. Finally, bold numbers indicate the best lower bound and upper bound on the value of the optimal solution for the respective instance.

Table 4.4: PCMCA-WT computational results of the linear relaxation of the models *MCF, CPB, RPB, DA* and *AAC* for TSPLIB instances.

Instance			Model																	
			MCF			CPB				RPB			DA				AAC			
Name	Size	$\rho(R)$	Cost	Time [s]	Gap	Cost	Cuts	Time [s]	Gap	Cost	Time [s]	Gap	Cost	Cuts	Time [s]	Gap	Cost	Cuts	Time [s]	Gap
br17.10	18	0.314	25.08	1.437	43.000	12.00	0	0.114	72.727	16.23	0.192	63.115	**25.17**	15	0.265	42.795	25.15	18	0.203	42.841
br17.12	18	0.359	25.12	1.032	42.909	17.00	11	0.136	61.364	16.23	0.224	63.115	**25.17**	15	0.265	42.795	25.15	18	0.203	42.841
ESC07	9	0.611	1887.50	0.204	0.971	1569.42	0	0.071	17.659	1612.21	0.061	15.414	**1890.75**	3	0.110	0.800	1782.07	7	0.031	6.502
ESC11	13	0.359	**2127.00**	0.297	2.162	2012.85	0	0.116	7.413	2061.27	0.114	5.186	2067.00	10	0.187	4.922	2040.30	8	0.312	6.150
ESC12	14	0.396	**1138.00**	0.109	0.000	**1138.00**	0	0.031	0.000	**1138.00**	0.021	0.000	**1138.00**	0	0.063	0.000	**1138.00**	1	0.078	0.000
ESC25	27	0.177	1043.05	3.297	9.927	**1091.04**	0	0.592	5.782	1078.72	0.315	6.846	1082.41	37	0.672	6.528	1064.20	40	0.890	8.100
ESC47	49	0.108	**703.14**	36.969	5.871	702.43	42	2.401	5.966	697.04	1.123	6.688	703.12	257	9.250	5.874	**703.14**	80	3.625	5.871
ESC63	65	0.173	**56.00**	266.610	0.000	**56.00**	0	2.030	0.000	**56.00**	2.024	0.000	**56.00**	6	1.594	0.000	**56.00**	67	20.937	0.000
ESC78	80	0.139	502.16	523.810	58.013	**794.29**	0	6.691	33.588	718.08	4.842	39.960	721.93	8	4.453	39.638	718.00	6	5.969	39.967
ft53.1	54	0.082	3953.05	188.391	3.325	3869.52	0	3.477	5.368	3887.09	3.084	4.938	**3962.45**	34	5.594	3.095	3949.66	25	8.297	3.408
ft53.2	54	0.094	3997.50	180.250	6.688	3922.54	0	4.251	8.437	3926.08	3.148	8.355	**3998.74**	40	5.531	6.659	3993.84	52	8.547	6.773
ft53.3	54	0.225	4286.90	171.203	20.006	4282.78	0	3.890	20.082	4228.38	3.246	21.098	**4388.35**	69	7.640	18.113	4249.72	97	13.562	20.699
ft53.4	54	0.604	5026.27	52.062	21.709	5123.04	0	3.282	20.202	5046.29	2.856	21.397	**5149.40**	18	4.875	19.791	5010.26	21	5.250	21.959
ft70.1	71	0.036	32801.04	1021.590	1.492	32588.06	0	4.900	2.132	32571.22	3.413	2.183	**32980.40**	148	16.610	0.954	32851.51	130	39.453	1.341
ft70.2	71	0.075	32895.06	1523.523	2.360	32601.02	0	3.537	3.232	32618.16	2.525	3.181	**33016.60**	160	22.235	1.999	32939.71	171	48.172	2.227
ft70.3	71	0.142	33441.93	2048.220	11.935	33074.18	0	5.055	12.903	32925.27	2.726	13.295	33641.84	402	47.500	11.408	**33672.54**	264	63.344	11.327
ft70.4	71	0.589	35433.67	113.969	11.533	34981.21	0	4.971	12.663	34837.52	3.817	13.021	**35805.55**	132	18.188	10.605	35427.98	156	31.813	11.547
rbg048a	50	0.444	**231.57**	335.875	11.951	224.60	0	2.925	14.601	225.74	1.604	14.169	228.06	11	1.703	13.285	221.84	11	3.985	15.650
rbg050c	52	0.459	215.12	124.781	4.391	**218.16**	0	1.502	3.040	214.13	2.233	4.833	214.35	36	2.485	4.733	217.24	26	3.422	3.449
rbg109	111	0.909	293.13	590.328	29.196	**324.47**	0	12.916	21.625	323.71	9.826	21.809	314.83	19	8.609	23.954	314.79	5	13.531	23.964
rbg150a	152	0.927	373.34	1417.090	27.927	**425.50**	0	21.793	17.858	423.14	9.936	18.314	417.14	12	10.969	19.471	416.17	7	32.969	19.658
rbg174a	176	0.929	365.40	2096.480	37.000	**414.52**	0	21.448	28.531	411.47	16.238	29.056	405.03	10	21.984	30.167	401.07	9	65.828	30.850
rbg253a	255	0.948	-	-	-	480.38	61	63.931	37.855	**484.67**	53.056	37.300	458.28	11	60.750	40.714	467.20	7	248.812	39.560
rbg323a	325	0.928	-	-	-	916.81	0	94.302	77.278	907.37	94.136	77.513	**920.95**	23	210.250	77.176	892.63	19	719.109	77.878
rbg341a	343	0.937	-	-	-	**722.76**	465	163.337	80.980	683.37	128.922	82.017	677.73	52	365.250	82.165	672.90	44	725.343	82.292
rbg358a	360	0.886	-	-	-	676.09	730	154.703	79.488	688.67	131.636	79.106	**699.25**	77	429.547	78.785	666.92	29	1395.735	79.766
rbg378a	380	0.894	-	-	-	-	-	-	-	607.47	186.618	77.982	**644.63**	107	422.203	76.635	605.73	61	1787.078	78.045
kro124p.1	101	0.046	32597.08	3482.940	7.476	32252.78	0	17.338	8.453	32212.80	10.305	8.567	**32657.90**	106	47.765	7.304	32603.69	123	89.266	7.457
kro124p.2	101	0.053	32761.06	3482.630	13.687	32574.70	0	17.972	14.178	32563.94	13.025	14.206	**33053.63**	135	48.688	12.916	32922.44	171	134.109	13.262
kro124p.3	101	0.092	33715.31	3490.750	34.000	32677.93	0	20.615	36.031	32852.59	17.095	35.689	**33951.74**	270	76.703	33.537	33826.66	303	212.000	33.782
kro124p.4	101	0.496	37386.23	2552.250	32.255	36774.18	0	16.331	33.364	36829.31	13.136	33.265	**38025.91**	132	35.250	31.096	37233.59	174	88.859	32.532
p43.1	44	0.101	**2825.00**	864.844	31.098	769.00	0	0.087	81.244	760.00	0.716	81.463	**2825.00**	49	2.140	31.098	2797.37	53	3.797	31.771
p43.2	44	0.126	2759.38	1036.547	33.268	880.00	0	0.800	78.718	970.00	1.220	76.542	**2825.00**	98	2.672	31.681	2722.91	140	9.422	34.150
p43.3	44	0.191	2759.53	573.968	43.452	1015.50	57	1.992	79.191	937.50	1.950	80.789	**2845.00**	113	3.469	41.701	2722.79	197	10.407	44.205
p43.4	44	0.164	2925.07	11.937	36.687	1229.82	20	1.774	73.380	1180.06	1.448	74.458	**2930.08**	115	2.984	36.578	2822.27	93	4.968	38.912
prob.100	100	0.048	643.00	3484.390	36.210	648.09	0	10.011	35.706	662.58	22.080	34.268	**668.13**	1225	598.594	33.717	657.65	1009	999.453	34.757
prob.42	42	0.116	148.90	57.672	12.924	150.45	0	1.380	12.018	150.03	1.821	12.264	**153.18**	107	4.813	10.421	148.26	52	5.469	13.298
ry48p.1	49	0.091	**13134.08**	99.141	4.285	12864.17	0	3.162	6.252	12842.68	2.000	6.408	13133.93	62	4.953	4.286	13115.36	54	8.391	4.421
ry48p.2	49	0.103	13195.09	95.937	8.943	12913.66	0	3.231	10.885	12906.53	2.669	10.934	**13243.77**	48	4.703	8.607	13206.48	34	5.203	8.864
ry48p.3	49	0.193	13926.14	136.859	11.904	13320.56	0	3.665	15.735	13236.22	2.747	16.269	**13979.71**	207	12.469	11.566	13925.41	191	19.016	11.909
ry48p.4	49	0.588	16168.48	16.781	16.735	15369.21	0	2.426	20.851	15399.42	1.800	20.695	**16316.13**	60	5.344	15.974	16186.84	93	9.406	16.640
Average				835.671	18.758		35	17.080	28.919		18.535	30.139		108	61.691	23.745		99	166.982	24.601

Table 4.5: PCMCA-WT computational results of the linear relaxation of the models *MCF, CPB, RPB, DA* and *AAC* for COMPILERS instances.

Instance			Model																	
			MCF			CPB				RPB			DA				AAC			
Name	Size	$\rho(R)$	Cost	Time [s]	Gap	Cost	Cuts	Time [s]	Gap	Cost	Time [s]	Gap	Cost	Cuts	Time [s]	Gap	Cost	Cuts	Time [s]	Gap
gsm.153.124	126	0.970	221.14	135.500	28.894	233.37	0	1.715	24.961	230.70	1.265	25.821	222.23	15	0.610	28.543	**223.41**	15	3.312	28.164
gsm.444.350	353	0.990	-	-	-	2195.86	0	16.133	23.569	**2237.31**	6.861	22.126	1914.83	6	4.531	33.351	2042.75	4	5.156	28.898
gsm.462.77	79	0.840	377.54	231.625	19.329	379.40	0	2.417	18.933	364.79	2.231	22.053	**384.41**	27	6.016	17.861	380.96	29	5.375	18.598
jpeg.1483.25	27	0.484	**84.00**	1.844	3.448	77.45	0	0.521	10.979	80.23	0.475	7.783	78.97	17	0.406	9.230	76.89	16	0.719	11.621
jpeg.3184.107	109	0.887	419.22	254.187	36.959	**453.39**	0	4.782	31.821	451.48	3.341	32.108	441.65	60	2.875	33.586	451.07	76	16.047	32.170
jpeg.3195.85	87	0.740	**13.04**	3595.590	47.840	9.26	137	1.890	62.964	8.98	1.566	64.083	9.00	126	7.875	64.000	13.00	195	9.130	48.000
jpeg.3198.93	95	0.752	140.26	3594.700	31.245	**161.30**	293	3.122	20.934	153.90	4.133	24.561	151.87	214	9.296	25.554	152.79	152	11.730	25.103
jpeg.3203.135	137	0.897	524.22	1217.125	30.104	**573.41**	0	7.197	23.546	566.77	4.948	24.431	564.03	58	3.234	24.796	568.97	122	21.063	24.137
jpeg.3740.15	17	0.257	**33.00**	0.313	0.000	31.00	0	0.100	6.061	31.00	0.091	6.061	**33.00**	5	0.093	0.000	**33.00**	3	0.125	0.000
jpeg.4154.36	38	0.633	**86.88**	7.843	3.467	82.00	0	0.801	8.889	81.01	0.475	9.988	85.06	26	2.125	5.489	84.01	21	0.765	6.656
jpeg.4753.54	56	0.769	150.20	76.875	8.415	**153.16**	0	1.386	6.609	149.40	0.915	8.899	153.08	30	3.500	6.659	150.19	40	2.250	8.421
susan.248.197	199	0.939	613.41	3519.028	48.192	683.20	0	7.412	42.297	**705.47**	3.715	40.416	658.84	108	9.672	44.355	682.70	138	34.656	42.340
susan.260.158	160	0.916	494.65	1681.391	43.533	526.81	0	7.326	39.861	524.74	5.916	40.098	519.01	116	7.796	40.752	**534.43**	244	55.359	38.992
susan.343.182	184	0.936	469.79	1488.110	43.399	549.14	0	5.902	33.839	**554.29**	4.246	33.218	539.47	72	6.156	35.004	554.04	94	20.234	33.248
typeset.10192.123	125	0.744	**246.52**	3579.440	40.598	258.30	0	8.340	37.758	259.05	5.594	37.579	**264.30**	131	22.078	36.313	260.60	90	23.328	37.205
typeset.10835.26	28	0.349	**93.55**	2.187	15.721	82.57	0	0.501	25.609	93.11	0.490	16.121	81.83	7	0.328	26.279	92.34	9	0.625	16.811
typeset.12395.43	45	0.518	**139.02**	57.437	4.781	133.04	0	0.794	8.880	133.04	0.797	8.875	137.85	110	3.094	5.582	137.27	107	4.484	5.979
typeset.15087.23	25	0.557	92.27	2.046	4.876	92.00	0	0.170	5.155	**93.00**	0.261	4.124	**93.00**	13	0.157	4.124	**93.00**	30	0.516	4.124
typeset.15577.36	38	0.555	120.01	4.141	3.992	**121.01**	0	0.642	3.196	121.00	0.566	3.199	120.69	21	0.531	3.448	**120.01**	14	1.015	3.992
typeset.16000.68	70	0.658	70.00	2051.330	12.500	63.00	71	1.940	21.250	61.00	0.784	23.750	**70.07**	121	4.062	12.413	69.48	486	32.735	13.150
typeset.1723.25	27	0.245	**56.00**	5.516	6.667	51.00	0	0.620	15.000	52.00	0.535	13.333	55.33	93	1.062	7.783	55.50	90	2.047	7.500
typeset.19972.246	248	0.993	-	-	-	1317.00	0	5.927	31.726	**1326.34**	2.227	31.242	1229.52	4	1.891	36.261	1234.43	11	4.328	36.007
typeset.4391.240	242	0.981	-	-	-	**1060.08**	0	8.852	24.923	1034.89	5.677	26.707	1006.12	31	2.812	28.745	1057.66	64	10.281	25.095
typeset.4597.45	47	0.493	**144.01**	23.484	7.090	141.03	0	1.230	9.012	141.98	1.142	8.398	143.13	89	3.282	7.658	141.18	169	9.281	8.916
typeset.4724.433	435	0.995	-	-	-	**2440.98**	0	20.665	28.897	2433.39	5.485	29.118	2351.03	29	12.016	31.517	2351.76	33	21.531	31.495
typeset.5797.33	35	0.748	105.93	2.843	6.257	105.00	0	0.431	7.082	104.60	0.315	7.434	**106.00**	9	0.625	6.195	104.21	21	0.891	7.779
typeset.5881.246	248	0.986	-	-	-	**1266.38**	0	5.192	25.507	1261.92	3.811	25.769	1204.29	27	5.234	29.159	1229.51	28	9.422	27.676
Average				978.753	20.332		19	4.297	22.195		2.513	22.122		58	4.495	22.395		85	11.348	21.188

Table 4.6: PCMCA-WT computational results of the linear relaxation of the models *MCF, CPB, RPB, DA* and *AAC* for SOPLIB instances.

Instance			Model															
			CPB				RPB			DA				AAC				
Name	Size	$\rho(R)$	Cost	Cuts	Time [s]	Gap	Cost	Time [s]	Gap	Cost	Cuts	Time [s]	Gap	Cost	Cuts	Time [s]	Gap	
R.200.100.1	200	0.020	**29.00**	0	17.648	0.000	**29.00**	24.012	0.000	**29.00**	7	20.078	0.000	**29.00**	11	110.821	0.000	
R.200.100.15	200	0.847	427.33	710	14.583	58.146	421.56	10.216	58.711	**482.25**	889	201.797	52.767	466.32	950	642.971	54.327	
R.200.100.30	200	0.957	571.47	399	15.301	69.891	541.45	8.875	71.473	**593.41**	61	21.703	68.735	580.69	58	30.857	69.405	
R.200.100.60	200	0.991	7105.17	0	22.940	60.952	7090.42	14.885	61.033	**7361.79**	0	17.750	59.542	7052.76	0	30.974	61.240	
R.200.1000.1	200	0.020	887.00	0	31.516	0.000	887.00	14.601	0.000	**887.00**	2	22.029	0.000	887.00	1	53.906	0.000	
R.200.1000.15	200	0.876	5702.41	464	11.143	65.432	5589.61	8.812	66.115	**6108.43**	490	144.953	62.970	6040.07	439	110.852	63.385	
R.200.1000.30	200	0.958	8199.45	168	15.640	66.114	8004.66	9.216	66.919	**8445.47**	23	14.281	65.097	8133.32	28	20.904	66.387	
R.200.1000.60	200	0.989	9511.48	0	25.658	56.515	9504.42	16.776	56.547	**9854.45**	0	22.125	54.947	9217.41	0	37.501	57.859	
R.300.100.1	300	0.013	**13.00**	0	37.862	0.000	**13.00**	42.470	0.000	**13.00**	16	23.625	0.000	**13.00**	2	636.572	0.000	
R.300.100.15	300	0.905	574.77	1716	31.446	83.759	543.25	16.375	84.649	**610.77**	239	105.640	82.742	609.37	422	266.235	82.781	
R.300.100.30	300	0.970	884.72	251	39.810	76.514	870.57	23.970	76.890	**903.36**	18	23.235	76.019	847.94	20	66.692	77.490	
R.300.100.60	300	0.994	750.82	0	58.215	69.441	769.16	45.044	68.695	**814.56**	13	45.469	66.848	750.10	306	409.733	69.471	
R.300.1000.1	300	0.013	715.00	0	172.601	0.000	715.00	125.470	0.000	**715.00**	9	179.203	0.000	715.00	13	621.532	0.000	
R.300.1000.15	300	0.905	6739.79	3859	33.435	71.354	6490.54	10.516	72.414	**7068.59**	186	87.157	69.957	**6793.90**	117	111.910	71.124	
R.300.1000.30	300	0.965	9905.82	1082	31.780	75.515	9848.70	20.710	75.656	**10157.39**	15	28.562	74.893	9718.07	18	68.687	75.979	
R.300.1000.60	300	0.994	8512.35	166	68.361	72.232	8463.01	42.675	72.393	**9084.57**	5	66.421	70.365	8337.93	3	76.061	72.801	
R.400.100.1	400	0.010	**6.00**	0	87.580	0.000	**6.00**	95.971	0.000	**6.00**	9	27.281	0.000	**6.00**	15	17.130	0.000	
R.400.100.15	400	0.927	679.92	2718	70.521	92.280	661.02	29.854	92.494	720.09	1335	122.000	91.824	**720.59**	679	689.301	91.818	
R.400.100.30	400	0.978	**797.69**	1182	103.335	88.987	763.91	42.011	89.453	**779.09**	33	35.220	89.244	758.85	26	75.670	89.523	
R.400.100.60	400	0.996	726.63	0	146.972	86.896	**734.00**	107.754	86.763	728.42	0	109.391	86.863	705.97	3	165.734	87.268	
R.400.1000.1	400	0.010	**780.00**	0	160.195	0.000	**780.00**	172.120	0.000	**780.00**	4	313.641	0.000	**780.00**	1	1734.093	0.000	
R.400.1000.15	400	0.930	7394.06	7119	72.876	91.390	7199.06	26.483	91.617	7591.79	294	99.682	91.160	**7645.09**	159	207.935	91.098	
R.400.1000.30	400	0.977	9698.42	708	65.620	89.847	9704.55	43.370	89.841	9896.44	51	53.220	89.640	**9935.48**	61	157.717	89.599	
R.400.1000.60	400	0.995	**8438.38**	0	141.884	84.918	7965.04	80.234	85.764	8098.89	1	62.223	85.525	7705.27	5	189.445	86.228	
R.500.100.1	500	0.008	**3.00**	0	149.530	0.000	**3.00**	163.800	0.000	**3.00**	23	57.094	0.000	**3.00**	172	2655.970	0.000	
R.500.100.15	500	0.945	869.55	3473	100.042	92.407	863.36	37.030	92.461	**913.85**	152	189.031	92.020	913.44	372	563.330	92.024	
R.500.100.30	500	0.980	751.87	537	106.410	93.850	751.05	76.237	93.856	744.60	31	67.500	93.909	**763.08**	43	172.505	93.758	
R.500.100.60	500	0.996	719.97	0	12.872	91.456	**751.04**	150.756	91.088	668.50	5	141.844	92.067	619.95	2	292.819	92.643	
R.500.1000.1	500	0.008	**297.00**	0	232.851	0.000	**297.00**	210.791	0.000	**297.00**	0	37.731	0.000	**297.00**	0	2493.358	0.000	
R.500.1000.15	500	0.940	8307.61	7467	107.512	92.292	8017.67	43.890	92.561	8295.58	116	99.593	92.303	**8401.04**	150	320.520	92.205	
R.500.1000.30	500	0.981	**10604.67**	2289	123.602	93.218	10191.19	54.800	93.482	10417.63	18	82.078	93.337	10181.27	14	150.795	93.489	
R.500.1000.60	500	0.996	7301.75	837	211.220	78.046	**7419.38**	217.383	77.693	7082.38	2	112.516	78.706	6857.44	5	425.744	79.382	
R.600.100.1	600	0.007	**1.00**	0	283.590	99.736	**1.00**	243.910	99.736	**1.00**	203	914.453	99.736	**1.00**	5	2131.340	99.736	
R.600.100.15	600	0.950	**682.84**	7593	148.082	88.522	659.76	86.221	88.910	642.09	54	116.891	89.207	668.95	81	333.780	88.755	
R.600.100.30	600	0.985	828.68	786	186.485	93.564	825.32	105.291	93.590	**843.46**	19	100.235	93.449	821.79	30	288.335	93.617	
R.600.100.60	600	0.997	743.32	0	415.432	90.583	**756.86**	281.033	90.411	736.24	0	182.078	90.672	677.55	1	703.480	91.416	
R.600.1000.1	600	0.007	**322.00**	0	486.591	0.000	**322.00**	445.056	0.000	**322.00**	0	65.987	0.000	**322.00**	0	368.367	0.000	
R.600.1000.15	600	0.945	**10180.92**	12384	186.856	91.647	9993.92	57.327	91.800	10180.54	99	178.547	91.647	10043.27	93	390.536	91.760	
R.600.1000.30	600	0.984	**10425.14**	1747	215.730	93.096	10380.56	103.210	93.126	10109.02	38	114.531	93.306	10017.22	40	365.387	93.367	
R.600.1000.60	600	0.997	**7935.40**	477	353.340	90.959	7878.47	303.187	91.024	7603.46	1	251.484	91.337	7366.04	9	962.126	91.608	
R.700.100.1	700	0.006	**2.00**	0	409.661	0.000	**2.00**	256.541	0.000	**2.00**	84	1770.352	0.000	**2.00**	153	2538.536	0.000	
R.700.100.15	700	0.957	**834.06**	7210	302.340	87.288	794.58	78.135	87.889	753.83	38	162.406	88.510	797.99	76	509.298	87.837	
R.700.100.30	700	0.987	**831.15**	659	355.100	95.902	732.60	297.645	96.388	719.89	9	140.344	96.450	760.97	11	531.510	96.248	
R.700.100.60	700	0.997	**537.21**	0	1491.985	92.360	535.36	1287.950	92.387	515.11	0	329.750	92.675	493.32	0	3119.500	92.985	
R.700.1000.1	700	0.006	**611.00**	0	1885.827	0.000	**611.00**	2135.741	0.000	**611.00**	0	121.523	0.000	**611.00**	0	2451.536	0.000	
R.700.1000.15	700	0.956	**4613.29**	5500	165.668	57.201	4579.24	125.875	57.517	4449.04	52	137.802	58.725	4586.78	12	434.986	57.447	
R.700.1000.30	700	0.986	**4377.11**	678	465.670	55.467	3921.48	204.601	60.103	4276.02	3	243.160	56.496	4295.71	1	498.860	56.296	
R.700.1000.60	700	0.997	2885.34	0	857.787	81.479	**2936.36**	859.320	81.152	2844.27	0	280.932	81.743	2845.67	0	3276.520	81.734	
Average				1504	223.565	62.901		184.545	63.179		97	161.345	62.613		96	677.341	63.002	

Table 4.7: PCMCA-WT computational results of the MILP models *MCF, CPB, RPB, DA* and *AAC* for TSPLIB instances.

Instance			Model																									
			MCF					CPB						RPB					DA						AAC					
Name	Size	$\rho(R)$	LB	UB	IP Gap	Nodes	Time [s]	LB	UB	IP Gap	Nodes	Cuts	Time [s]	LB	UB	IP Gap	Nodes	Time [s]	LB	UB	IP Gap	Nodes	Cuts	Time [s]	LB	UB	IP Gap	Nodes	Cuts	Time [s]
br17.10	18	0.314	**35**	**44**	20.455	192505	-	31	**44**	29.545	3276683	30	-	**35**	**44**	20.455	1247254	-	34	44	22.727	2767833	3799	-	34	**44**	22.727	2074173	3893	-
br17.12	18	0.359	35	**44**	20.455	291653	-	35	**44**	20.455	3717500	34	-	**38**	**44**	13.636	1286779	-	35	45	22.222	3376744	2519	-	35	**44**	20.455	2845672	2265	-
ESC07	9	0.611	1906	1906	0.000	7	0.109	1906	1906	0.000	17	0	0.081	1906	1906	0.000	16	0.071	1906	1906	0.000	17	5	0.078	1906	1906	0.000	0	3	0.078
ESC11	13	0.359	2174	2174	0.000	20	0.265	2174	2174	0.000	250	0	0.555	2174	2174	0.000	72	0.236	2174	2174	0.000	22	13	0.172	2174	2174	0.000	33	12	0.219
ESC12	14	0.396	1138	1138	0.000	0	0.078	1138	1138	0.000	0	0	0.031	1138	1138	0.000	0	0.021	1138	1138	0.000	0	0	0.063	1138	1138	0.000	0	1	0.078
ESC25	27	0.177	1158	1158	0.000	1158	26.625	1158	1158	0.000	1302	0	17.351	1158	1158	0.000	1399	6.652	1158	1158	0.000	1450	698	3.922	1158	1158	0.000	1098	452	4.812
ESC47	49	0.108	747	747	0.000	2776	3154.140	747	747	0.000	10129	42	600.111	747	747	0.000	10248	99.119	747	747	0.000	750	31743	2230.171	704	783	10.089	2	37495	-
ESC63	65	0.173	56	61	8.197	165	-	56	56	0.000	3000	0	329.581	56	56	0.000	15325	293.443	56	56	0.000	4100	287	53.281	56	56	0.000	3177	14118	1223.313
ESC78	80	0.139	922	1346	31.501	920	-	1044	1196	12.709	24186	0	-	1196	1196	0.000	484412	1969.757	1196	1196	0.000	11177	267	159.093	1196	1196	0.000	15734	943	301.625
ft53.1	54	0.082	3984	4089	2.568	790	-	3989	4190	4.797	28963	0	-	4033	4089	1.370	162065	-	4089	4089	0.000	134416	2932	1430.797	4089	4089	0.000	56905	1433	1195.875
ft53.2	54	0.094	4054	4659	12.986	1055	-	4087	4439	7.930	28421	0	-	4134	4509	8.317	68193	-	**4135**	**4284**	3.478	200163	6053	-	4112	4318	4.771	57840	8393	-
ft53.3	54	0.225	4472	6071	26.338	1030	-	4585	5491	16.500	29027	0	-	**4729**	**5359**	11.756	147869	-	4623	5457	15.283	168357	6749	-	4545	5734	20.736	92513	5778	-
ft53.4	54	0.604	5409	6871	21.278	6531	-	5756	6561	12.269	45600	0	-	**5835**	**6420**	9.112	324443	-	5657	6439	12.145	564462	2361	-	5559	6668	16.632	340573	1588	-
ft70.1	71	0.036	32937	37078	11.168	807	-	32737	33308	1.714	13726	0	-	32779	33519	2.208	18918	-	33117	**33298**	0.544	23240	11362	-	**33128**	33714	1.738	27834	6322	-
ft70.2	71	0.075	33084	43282	23.562	408	-	32951	35076	6.058	12497	0	-	33036	**33690**	1.941	36056	-	**33357**	34450	3.173	11000	13238	-	33297	34713	4.079	11474	10171	-
ft70.3	71	0.142	33693	66153	49.068	287	-	33888	38744	12.534	13008	0	-	**34298**	**37974**	9.680	44598	-	33914	42732	20.636	8574	15867	-	33830	49116	31.122	9040	11611	-
ft70.4	71	0.589	35912	44507	19.312	1969	-	36445	**40053**	9.008	20400	0	-	**36538**	40248	9.218	141064	-	36517	40404	9.620	121796	3615	-	36188	40701	11.088	78814	3008	-
rbg048a	50	0.444	259	267	2.996	596	-	259	264	1.894	49765	42	-	260	263	1.141	285696	-	**261**	**263**	0.760	666476	15025	-	260	266	2.256	62610	14794	-
rbg050c	52	0.459	225	236	4.661	847	-	225	225	0.000	494	0	39.219	225	225	0.000	2210	78.845	225	225	0.000	1260	326	10.469	225	225	0.000	24050	432	210.562
rbg109	111	0.909	320	699	54.220	529	-	353	437	19.222	14472	0	-	**362**	430	15.814	86695	-	354	**414**	14.493	150506	219	-	349	428	18.458	85685	601	-
rbg150a	152	0.927	403	972	58.539	384	-	452	563	19.716	6819	0	-	**456**	**518**	11.969	33910	-	447	541	17.375	59431	232	-	441	650	32.154	34702	618	-
rbg174a	176	0.929	400	1121	64.318	326	-	452	678	33.333	4408	177	-	**461**	606	23.927	23056	-	446	**580**	23.103	42286	302	-	441	627	29.665	24226	113	-
rbg253a	255	0.948	-	-	-	-	-	517	1446	64.246	1402	61	-	**529**	1033	48.790	9502	-	477	**773**	38.292	12700	152	-	475	971	51.081	5427	231	-
rbg323a	325	0.928	-	-	-	-	-	**969**	**4035**	75.985	616	0	-	930	**4035**	76.952	779	-	926	4367	78.796	4536	86	-	903	**4035**	77.621	2845	78	-
rbg341a	343	0.937	-	-	-	-	-	**758**	5390	85.937	369	465	-	727	**3800**	80.868	1094	-	681	3832	82.229	3963	116	-	680	**3800**	82.105	2300	90	-
rbg358a	360	0.886	-	-	-	-	-	**717**	11088	93.534	153	1695	-	694	4109	83.110	416	-	706	**3296**	78.580	3408	698	-	674	4109	83.597	1779	144	-
rbg378a	380	0.894	-	-	-	-	-	-	-	-	-	-	-	624	4112	84.825	411	-	**649**	**2759**	76.477	2831	110	-	613	6701	90.852	1128	106	-
kro124p.1	101	0.046	32598	45652	28.595	1	-	32510	38880	16.384	4697	0	-	32545	40370	19.383	3343	-	**32858**	35438	7.280	3389	139	-	32827	**35231**	6.824	6926	5724	-
kro124p.2	101	0.053	32762	55265	40.718	1	-	32822	42140	22.112	4549	0	-	32950	48188	31.622	6840	-	**33190**	**37956**	12.557	14237	5920	-	33163	38002	12.734	8410	3458	-
kro124p.3	101	0.092	33716	329579	89.770	1	-	33337	63394	47.413	4595	0	-	33739	**51084**	33.954	10938	-	**34217**	53988	36.621	20016	6098	-	33991	77266	56.008	3313	9747	-
kro124p.4	101	0.496	37395	103491	63.866	2	-	38512	63496	39.347	7376	0	-	38856	56026	30.646	55450	-	**39413**	**55187**	28.583	71246	2652	-	38269	62162	38.437	32120	2746	-
p43.1	44	0.101	2825	4470	36.801	779	-	904	4775	81.068	54093	0	-	2660	**4100**	35.122	339789	-	**2827**	4585	38.342	160185	19944	-	2825	4615	38.787	51091	11280	-
p43.2	44	0.126	2760	4845	43.034	430	-	941	5025	81.274	50209	61	-	1475	**4135**	64.329	437665	-	**2826**	4275	33.895	162580	18401	-	2826	5025	43.761	55765	15014	-
p43.3	44	0.191	2828	6955	59.339	511	-	1056	5280	80.000	54587	129	-	1126	**4880**	76.926	541080	-	**2864**	6105	53.088	85658	16311	-	2847	5375	47.033	26582	10567	-
p43.4	44	0.164	2991	5415	44.765	5826	-	3001	4985	39.799	93774	280	-	3073	**4620**	33.485	531258	-	**3101**	4900	36.714	875030	2702	-	2981	4970	40.020	408540	1620	-
prob.100	100	0.048	643	3611	82.193	1	-	665	1055	36.967	4582	0	-	669	1053	36.467	5123	-	**674**	1047	35.626	3718	9961	-	661	**1008**	34.425	1533	7323	-
prob.42	42	0.116	163	171	4.678	2746	-	168	171	1.754	73225	0	-	171	171	0.000	442628	869.732	171	171	0.000	112468	7561	945.203	167	171	2.339	157734	11472	-
ry48p.1	49	0.091	13278	15362	13.566	1339	-	13038	15762	17.282	29428	0	-	13119	14413	8.978	29480	-	13356	13854	3.595	158744	8620	-	**13371**	**13722**	2.558	85005	6664	-
ry48p.2	49	0.103	13403	17833	24.842	585	-	13135	14734	10.852	29191	0	-	13243	**14491**	8.612	64977	-	13507	16968	20.397	82990	10129	-	**13508**	14659	7.852	63149	6161	-
ry48p.3	49	0.193	13984	21379	34.590	773	-	13598	16173	15.922	32347	0	-	13828	**15808**	12.525	90705	-	**14371**	16337	12.034	70000	8508	-	14299	16326	12.416	38332	6103	-
ry48p.4	49	0.588	16922	20030	15.517	10691	-	16664	**19418**	14.183	55351	0	-	16939	19444	12.883	245943	-	**17339**	19649	11.756	699363	921	-	17136	20915	18.068	398622	992	-
Average					28.164	14679	636.243			25.794	195030	75	140.990			22.440	176529	368.653			20.742	264905	5772	483.325			23.719	175531	5453	367.070

Table 4.8: PCMCA-WT computational results of the MILP models *MCF, CPB, RPB, DA* and *AAC* for COMPILERS instances.

Instance			Model																									
			MCF					CPB						RPB					DA						AAC					
Name	Size	$\rho(R)$	LB	UB	IP Gap	Nodes	Time [s]	LB	UB	IP Gap	Nodes	Cuts	Time [s]	LB	UB	IP Gap	Nodes	Time [s]	LB	UB	IP Gap	Nodes	Cuts	Time [s]	LB	UB	IP Gap	Nodes	Cuts	Time [s]
gsm.153.124	126	0.970	234	331	29.305	936	-	255	312	18.269	142388	0	-	**256**	**311**	17.685	862140	-	246	313	21.406	357665	413	-	243	319	23.824	270962	405	-
gsm.444.350	353	0.990	-	-	-	-	-	2334	3353	30.391	20400	0	-	**2406**	4896	50.858	21364	-	1996	**2873**	30.526	50228	209	-	2103	2905	27.608	27606	216	-
gsm.462.77	79	0.840	392	707	44.554	1150	-	400	479	16.493	82005	0	-	**408**	**468**	12.821	355976	-	396	493	19.675	462000	1734	-	391	488	19.877	185939	2399	-
jpeg.1483.25	27	0.484	87	87	0.000	2338	71.610	87	87	0.000	17000	0	161.464	87	87	0.000	1943	22.499	87	87	0.000	20708	583	11.254	87	87	0.000	33035	553	72.907
jpeg.3184.107	109	0.887	430	811	46.979	992	-	504	665	24.211	55602	0	-	**508**	859	40.861	341998	-	488	684	28.655	315065	1051	-	489	684	28.509	121292	744	-
jpeg.3195.85	87	0.740	14	76	81.579	0	-	14	187	92.513	70215	5008	-	17	25	32.000	377784	-	**22**	**25**	12.000	27085	9849	-	21	30	30.000	2955	3273	-
jpeg.3198.93	95	0.752	141	353	60.057	0	-	**180**	248	27.419	51600	5218	-	179	249	28.112	41417	-	172	213	19.249	89431	2584	-	161	**204**	21.078	16822	3448	-
jpeg.3203.135	137	0.897	535	1539	65.237	14	-	607	840	27.738	30808	0	-	**616**	841	26.754	245218	-	600	755	20.530	179785	2533	-	595	**750**	20.667	120417	1332	-
jpeg.3740.15	17	0.257	33	33	0.000	0	0.266	33	33	0.000	692	0	2.648	33	33	0.000	316	1.335	33	33	0.000	0	5	0.093	33	33	0.000	0	3	0.125
jpeg.4154.36	38	0.633	90	90	0.000	10271	139.579	90	90	0.000	303926	0	1459.161	90	90	0.000	31852	114.724	90	90	0.000	15705	461	25.766	90	90	0.000	4242	298	12.562
jpeg.4753.54	56	0.769	157	174	9.770	1826	-	161	164	1.829	295042	0	-	163	165	1.212	965367	-	163	165	1.212	1128020	1079	-	164	164	0.000	594353	911	2231.235
susan.248.197	199	0.939	614	3014	79.628	0	-	791	1648	52.002	23456	0	-	**804**	1660	51.566	51785	-	718	**1184**	39.358	62265	1519	-	736	1353	45.602	39582	1511	
susan.260.158	160	0.916	498	2530	80.316	74	-	566	1046	45.889	25041	0	-	**576**	988	41.700	85169	-	541	1149	52.916	163071	3999	-	564	**876**	35.616	49000	1510	-
susan.343.182	184	0.936	527	1433	63.224	5	-	612	**830**	26.265	28313	0	-	**621**	855	27.368	169618	-	586	862	32.019	111339	901	-	591	887	33.371	55306	1143	-
typeset.10192.123	125	0.744	247	774	68.088	0	-	272	473	42.495	16827	56	-	278	423	34.279	113747	-	**280**	**415**	32.530	92153	3643	-	**280**	456	38.596	59000	1924	-
typeset.10835.26	28	0.349	99	114	13.158	119310	-	98	112	12.500	1623454	0	-	**99**	**111**	10.811	697710	-	**99**	112	11.607	986681	7100	-	**99**	113	12.389	577573	5961	-
typeset.12395.43	45	0.518	141	148	4.730	5556	-	139	**146**	4.795	83710	0	-	139	**146**	4.795	374260	-	**143**	**146**	2.055	392093	4782	-	141	147	4.082	175106	4776	-
typeset.15087.23	25	0.557	97	97	0.000	10917	-	97	97	0.000	159259	0	366.055	97	97	0.000	31637	69.139	97	97	0.000	24721	1082	29.235	97	97	0.000	13225	759	32.094
typeset.15577.36	38	0.555	125	125	0.000	12472	1738.422	125	125	0.000	64560	0	1259.265	125	125	0.000	18210	106.197	125	125	0.000	18600	1467	51.688	125	125	0.000	106552	3042	834.797
typeset.16000.68	70	0.658	71	102	30.392	2	-	66	82	19.512	82175	2119	-	66	**80**	17.500	446480	-	**77**	86	10.465	13917	28566	-	**77**	**80**	3.750	29319	6506	-
typeset.1723.25	27	0.245	60	60	0.000	682	84.750	60	60	0.000	534739	0	2118.056	60	60	0.000	67379	203.365	60	60	0.000	1212	281	3.391	60	60	0.000	5533	481	29.734
typeset.19972.246	248	0.993	-	-	-	-	-	1420	3313	57.139	39254	0	-	**1427**	2898	50.759	275575	-	1325	1963	32.501	68601	48	-	1307	**1929**	32.245	33530	45	-
typeset.4391.240	242	0.981	-	-	-	-	-	1122	2786	59.727	33219	0	-	**1132**	2476	54.281	118813	-	1067	1419	24.806	100406	332	-	1093	**1412**	22.592	43577	388	-
typeset.4597.45	47	0.493	147	167	11.976	3247	-	149	**155**	3.871	135384	0	-	149	156	4.487	421938	-	**150**	**155**	3.226	501643	4681	-	149	159	6.289	107981	3505	-
typeset.4724.433	435	0.995	-	-	-	-	-	**2576**	7013	63.268	16285	0	-	2537	5493	53.814	118853	-	2378	5376	55.766	29538	214	-	2460	**3433**	28.343	15907	211	-
typeset.5797.33	35	0.748	113	113	0.000	2872	100.234	113	113	0.000	342279	0	-	113	113	0.000	20506	59.900	113	113	0.000	15172	652	24.250	113	113	0.000	10733	341	24.953
typeset.5881.246	248	0.986	-	-	-	-	-	1393	2232	37.590	36700	0	-	**1411**	2054	31.305	60603	-	1258	1877	32.978	105414	318	-	1305	**1700**	23.235	44716	183	-
Average					31.318	7848	355.810			24.589	159790	459	894.441			21.962	233987	82.451			17.907	197501	2966	20.811			16.951	101639	1699	404.801

Table 4.9: PCMCA-WT computational results of the MILP models *MCF, CPB, RPB, DA* and *AAC* for SOPLIB instances.

Instance			Model																					
			CPB						RPB					DA						AAC					
Name	Size	$\rho(R)$	LB	UB	IP Gap	Nodes	Cuts	Time [s]	LB	UB	IP Gap	Nodes	Time [s]	LB	UB	IP Gap	Nodes	Cuts	Time [s]	LB	UB	IP Gap	Nodes	Cuts	Time [s]
R.200.100.1	200	0.020	29	29	0.000	1768	0	3213.574	29	29	0.000	0	24.012	29	29	0.000	1451	3261	848.387	29	29	0.000	50	74	139.643
R.200.100.15	200	0.847	497	1746	71.535	7842	1805	-	**527**	**1021**	48.384	53119	-	499	2080	76.010	33850	2316	-	505	1271	60.268	23601	3229	-
R.200.100.30	200	0.957	697	2763	74.774	15201	1170	-	**787**	**1898**	58.535	17855	-	650	2485	73.843	79488	358	-	669	2011	66.733	65134	393	-
R.200.100.60	200	0.991	8362	18659	55.185	7667	0	-	**8649**	**18196**	52.468	10968	-	8070	18990	57.504	61665	13	-	7938	18761	57.689	48348	10	-
R.200.1000.1	200	0.020	887	887	0.000	0	0	31.516	887	887	0.000	0	14.601	887	887	0.000	0	2	22.029	887	887	0.000	0	1	53.906
R.200.1000.15	200	0.876	6595	21624	69.501	9253	1876	-	**6975**	22059	68.380	10893	-	6665	**16496**	59.596	40385	1638	-	6507	16912	61.524	35789	1527	-
R.200.1000.30	200	0.958	9845	25261	61.027	13673	168	-	**10369**	**24197**	57.148	12336	-	9340	30664	69.541	82262	328	-	8841	30351	70.871	64647	258	-
R.200.1000.60	200	0.989	11320	24785	54.327	7800	80	-	**11612**	**21873**	46.912	11980	-	10508	28507	63.139	64769	21	-	10364	23748	56.358	29186	23	-
R.300.100.1	300	0.013	13	68	80.882	809	0	-	13	13	0.000	487	537.607	13	13	0.000	200	73	211.776	13	13	0.000	200	576	993.472
R.300.100.15	300	0.905	645	6334	89.817	4837	5196	-	**661**	**3539**	81.322	15553	-	625	15979	96.089	23389	1657	-	619	12903	95.203	19199	1646	-
R.300.100.30	300	0.970	1008	4737	78.721	5352	510	-	**1075**	4179	74.276	3168	-	948	7458	87.289	29523	483	-	889	**3767**	76.400	26005	186	-
R.300.100.60	300	0.994	895	5080	82.382	3400	0	-	**919**	**2457**	62.597	3697	-	824	5013	83.563	21330	129	-	804	3005	73.245	12659	486	-
R.300.1000.1	300	0.013	715	715	0.000	0	0	172.601	715	715	0.000	47	1061.380	715	715	0.000	1032	1809	1889.116	715	719	0.556	308	4592	-
R.300.1000.15	300	0.905	7435	58340	87.256	4827	6870	-	**7597**	**23528**	67.711	24212	-	7213	134017	94.618	17525	1599	-	7013	112424	93.762	17304	1452	-
R.300.1000.30	300	0.965	11119	88485	87.434	4600	1204	-	**11457**	42659	73.143	11512	-	10385	85785	87.894	26988	277	-	10255	**40457**	74.652	17698	210	-
R.300.1000.60	300	0.994	9815	65180	84.942	2400	166	-	**10061**	34144	70.534	3970	-	9413	33198	71.646	20904	15	-	9031	**30655**	70.540	15933	91	-
R.400.100.1	400	0.010	6	376	98.404	231	0	-	6	6	0.000	274	2623.634	6	6	0.000	72	52	162.866	6	6	0.000	24	39	2474.545
R.400.100.15	400	0.927	757	38671	98.042	2436	9344	-	**836**	**8807**	90.508	2341	-	722	47117	98.468	10860	810	-	729	47117	98.453	6714	1070	-
R.400.100.30	400	0.978	890	39022	97.719	2279	2373	-	**976**	12693	92.311	1409	-	780	9434	91.732	14841	128	-	760	**7243**	89.507	11406	112	-
R.400.100.60	400	0.996	807	17435	95.371	1298	0	-	**847**	8319	89.818	2310	-	731	**5545**	86.817	9144	5	-	709	17435	95.933	6586	2	-
R.400.1000.1	400	0.010	780	12643	93.831	47	0	-	780	10025	92.219	60	-	780	780	0.000	0	6	239.929	780	780	0.000	0	1	1734.093
R.400.1000.15	400	0.930	7986	120315	93.362	2614	12923	-	**8545**	**85878**	90.050	3956	-	7647	501543	98.475	11229	1013	-	7760	501543	98.453	8643	809	-
R.400.1000.30	400	0.977	10665	267620	96.015	2209	1490	-	**11910**	127290	90.643	761	-	9927	97441	89.812	13546	183	-	10076	**95523**	89.452	10687	206	-
R.400.1000.60	400	0.995	9185	212880	95.685	1005	0	-	**10199**	160443	93.643	598	-	8103	68205	88.120	8847	7	-	7933	**55950**	85.821	6653	6	-
R.500.100.1	500	0.008	3	384	99.219	100	0	-	3	276	98.913	109	-	3	3	0.000	5	172	233.564	3	3	0.000	10	288	2837.404
R.500.100.15	500	0.945	947	21558	95.607	1408	13214	-	**1011**	**11452**	91.172	2816	-	924	81226	98.862	6785	519	-	919	11452	91.975	3610	799	-
R.500.100.30	500	0.980	843	19642	95.708	1661	2897	-	**980**	14723	93.344	587	-	745	**12225**	93.906	7690	51	-	773	21749	96.446	5650	111	-
R.500.100.60	500	0.996	**811**	22047	96.321	0	245	-	771	21032	96.334	658	-	669	**8427**	92.061	4367	5	-	629	21032	97.009	3294	9	-
R.500.1000.1	500	0.008	297	299	0.669	64	0	-	297	297	0.000	34	1646.009	297	297	0.000	0	0	37.731	297	297	0.000	0	0	2493.358
R.500.1000.15	500	0.940	**8831**	138150	93.608	1384	14583	-	8825	**107776**	91.812	3923	-	8297	667814	98.758	5092	560	-	8420	107776	92.188	4280	463	-
R.500.1000.30	500	0.981	11555	**156359**	92.610	1147	2647	-	**12419**	**156359**	92.057	488	-	10431	213239	95.108	6719	138	-	10182	181835	94.400	5880	91	-
R.500.1000.60	500	0.996	7806	43878	82.210	441	837	-	**8770**	43878	80.013	610	-	7094	104889	93.237	4231	6	-	6929	**33260**	79.167	3348	11	-
R.600.100.1	600	0.007	1	664	99.849	63	0	-	**1**	**379**	99.736	54	-	1	**379**	99.736	5	1161	-	**1**	**379**	99.736	2	79	-
R.600.100.15	600	0.950	743	13272	94.402	702	17651	-	**831**	13272	93.739	530	-	646	**5949**	89.141	4985	222	-	670	13272	94.952	3743	223	-
R.600.100.30	600	0.985	905	18932	95.220	658	1350	-	**1106**	18932	94.158	482	-	873	**12875**	93.219	4591	58	-	822	18932	95.658	3216	78	-
R.600.100.60	600	0.997	**805**	26732	96.989	313	16	-	782	26732	97.075	427	-	751	7893	90.485	2696	0	-	694	26732	97.404	1234	1	-
R.600.1000.1	600	0.007	322	322	0.000	0	0	486.591	322	322	0.000	0	445.056	322	322	0.000	0	0	65.987	322	322	0.000	0	0	368.367
R.600.1000.15	600	0.945	**10600**	**121877**	91.303	434	19985	-	10464	**121877**	91.414	542	-	10181	899457	98.868	3120	576	-	10067	121877	91.740	2093	502	-
R.600.1000.30	600	0.984	11252	190145	94.082	194	4117	-	**11668**	190145	93.864	525	-	10151	**151010**	93.278	4124	90	-	10088	190145	94.695	3195	93	-
R.600.1000.60	600	0.997	7966	256464	96.894	10	966	-	**8027**	256464	96.870	251	-	7604	96359	92.109	1952	12	-	7468	**87770**	91.491	1109	18	-
R.700.100.1	700	0.006	2	447	99.553	24	0	-	2	447	99.553	40	-	2	2	0.000	3	196	1788.796	**2**	**2**	0.000	**5**	**273**	3216.253
R.700.100.15	700	0.957	877	14478	93.943	130	11401	-	**962**	6976	86.210	483	-	768	**6561**	88.294	3262	117	-	799	6977	88.548	2432	207	-
R.700.100.30	700	0.987	**843**	22673	96.282	709	22	-	819	22673	96.388	520	-	721	**20281**	96.445	3180	29	-	762	22673	96.639	2125	35	-
R.700.100.60	700	0.997	544	20083	97.291	16	8	-	**567**	**7032**	91.937	423	-	516	9030	94.286	1627	0	-	498	27116	98.163	10	0	-
R.700.1000.1	700	0.006	611	631	3.170	4	22	-	611	611	0.000	0	2135.741	611	621	1.610	231	8	-	611	1657	63.126	4	26	-
R.700.1000.15	700	0.956	4844	147321	96.712	394	6364	-	**5075**	**10779**	52.918	718	-	4456	147321	96.975	4043	153	-	4636	147321	96.853	2365	60	-
R.700.1000.30	700	0.986	4546	208461	97.819	108	678	-	**4777**	**9829**	51.399	510	-	4277	252128	98.304	2848	19	-	4303	50000	91.394	1677	11	-
R.700.1000.60	700	0.997	2901	277619	98.955	9	0	-	**3004**	277619	98.918	178	-	2855	29769	90.409	1236	1	-	2857	**15579**	81.661	3	0	-
Average					78.221	2323	2962	976.071			68.509	4300	1061.005			68.109	13460	422	550.018			67.681	9918	425	1590.116

4.6.2 The CP Models

In this section we evaluate and discuss the performance of the CP models proposed in section 4.5, and compare them with the *Path-Based* models proposed in section 4.4.2 that the CP models are based on.

For the rest of this section we will be referring to the *Complete Path-Based* model introduced in section 4.4.2.1 as *CPB*, the *Reduced Path-Based* model introduced in section 4.4.2.2 as *RPB*, the *Complete CP* model introduced in section 4.5.1 as *CCP*, and the *Reduced CP* model introduced in section 4.5.2 as *RCP*.

All the experiments in this section are performed on an Intel Xeon Platinum 8375C processor with 8 cores running at 2.9 GHz with 16 GB of RAM. The two *Path-Based* MILP models are solved using CPLEX v12.8 [53], with 8 thread standard B&C algorithm is applied for solving the models, and the two parameters *NodeSelect* and *MIP emphasis* are set to *BestBound* and *MIPEmphasisOptimality* respectively. On the other hand, the two CP models are solved using Google OR-Tools [46] v9.5 CP-SAT solver (preliminary results have shown that CPLEX CP solver performs considerably worse compared to OR-Tools CP-SAT solver), with all 8 threads available are allocated for the solver. A time limit of 1 hour is set on the computation time of both solvers, where the preprocessing time required to compute zero-cost paths for the *Reduced Path-Based* and *Reduced CP* model are not considered, being negligible for all the instances considered. The best-known solutions are obtained from the results appeared in tables 4.7-4.9 (containing 87 open instances), that were performed on an Intel i7-8550U processor running at 1.8 GHz with 8 GB of RAM. CPLEX v12.8 is used for solving the models, configured with the same parameters mentioned previously, but a single thread is allocated for the solver, and a time limit of 1 hour is set on the computation time of the solver. The comparison between the computational results introduced in this section and

Table 4.10: PCMCA-WT summary of the results achieved by the MILP and CP Solvers for the models *CPB, RPB, CCP,* and *RCP*.

	Model			
	CPB	RPB	CCP	RCP
Average optimality gap	20.4%	15.3%	15.7%	12.0%
Average solution time	690.5	297.6	187.1	38.7
New best-known lower bounds	3	21	15	31
New best-known upper bounds	2	12	6	51
New optimal solutions	0	0	6	6

the best-known results is not fair, however preliminary results have shown that the *Path-Based* models (more specifically the *Reduced Path-Based* model) outperform the rest of the models introduced in section 4.4 on average even when 8 threads are allocated for the solver.

Table 4.10 summarizes the performance of each model in terms of solution time, and the quality of the obtained solutions. In the table, the average optimality gap reports the average optimality gap for all the instances where all four models are able to find a feasible/optimal solution before reaching the time limit (a total of 79 out of 116 instances). The average solution time reports the average solution time in seconds of all the instances that are optimally solved by all four models within the time limit (a total of 27 out of 116 instances).

Comparing the average optimality gap of each model, the model *CPB* achieves an average optimality gap of 20.4% (41.7% across 114 instances), but fails to solve two instance (*R.700.100.1* and *typeset.4724.433*) as it runs out of memory while solving the linear relaxation of the model. The model *RPB* achieves an average optimality gap of 15.3% (a 25% improvement), and an average optimality gap of 33.8% (a 18.9% improvement across 114 instances) when excluding the instance

that is not solved by the model *CPB*, and an average optimality gap of 34.0% (an 18.7% improvement) across all the instances. The model *RPB* also fails to solve a single instance (*R.700.100.1*) as it runs out of memory while solving the linear relaxation of the model. On the other hand, the model *CCP* achieves an average optimality gap of 15.7% (a 23.0% improvement), but fails to solve the instances with size larger than 200 with a very dense precedence graph, as it runs out of memory while building the model due to the large number of constraints (4.26). Finally, the model *RCP* achieves an average optimality gap of 12.0% (a 41.2% improvement), and an average optimality gap of 28.6% across all the instances.

In terms of solution time, and comparing the instances that are optimally solved by all models (27 instances), the model *CPB* has an average solution time of 690.5 seconds, while the model *RPB* has an average solution time of 297.6 seconds (a 56.9% improvement). On the other hand, the model *CCP* has an average solution time of 187.1 seconds (a 72.9% improvement), while the model *RCP* has an average solution time of 38.7 seconds (a 94.4% improvement). We should note that the model *RPB* generally finds the optimal solution in less time compared to the model *CPB*, however there is no clear pattern which instances fall in that criteria. On the other hand, the model *RCP* finds the optimal solution in less time compared to the model *CCP* on instances with low to medium density precedence graphs. Finally, comparing the MILP and CP models, the CP models generally outperform the MILP models on instances with medium to high density precedence graphs.

In terms of solution costs, the model *RPB* finds new best-known lower bounds for 21 out of 87 instances (24.1%) compared to the model *CPB* which finds a new best-known lower bound for 3 out of 87 instances (3.5%). Furthermore, the model *RPB* finds new best-known upper bounds for 12 out of 87 instances (13.8%) compared to the model *CPB* which finds a new best-known upper bound for 2 out of 87 instances (2.3%). This indicates that the strength of the linear relaxation of the model is not

drastically affected after removing a subset of the variables and constraints from the model. Furthermore, this shows that the model *RPB* is generally easier to solve, and therefore the solver is able to find new bounds more frequently compared to solving the model *CPB*. On the other hand, the model *RCP* finds new best-known lower bounds for 31 out of 87 instances (35.6%) compared to the model *CCP* which finds new lower bounds for 15 out of 87 instances (17.2%). Furthermore, the model *RCP* finds new best-known upper bounds for 51 out of 87 instances (58.6%) compared to the model *CCP* which finds a new best-known upper bound for 6 out of 87 instances (6.9%). This indicates that the model *RCP* is generally more effective compared to the model *CCP*. Moreover, the two proposed CP models are able to find the optimal solution for 6 out of 87 open instances. Generally, the MILP models are better at finding tighter estimates on the lower bounds (when accounting for previous results), while the CP models are better at finding lower cost solutions.

In summary, the computational results shows that the *reduced* models have an overall better performance compared to the *complete* models, in both average solution time, achieved average optimality gap, and solution cost. This is due to the fact that the *reduced* models have a smaller number of variables and constraints, which makes them much easier to solve in theory compared to the *complete* models. More specifically, the CP models outperform the MILP models on average, in both the quality of the solutions, the average solution time, and average optimality gap. Furthermore, the CP models find new best-known lower/upper bounds for a larger number of instances. The complete results of each model can be found in tables 4.11-4.13, where we report the following for each instance. Columns *Name* and *Size* report the name and size of the instance. Column $\rho(R)$ reports the density of arcs in the set of precedence relationships computed as $\frac{2 \cdot |R|}{|V|(|V|-1)}$. Column *Best-Known* reports the best-known bounds on the optimal solution for each instance as $[LB, UB]$, where LB is the lower bound on the optimal solution, and UB is the best-known

solution. For each model solved, we report the following columns. Columns *LB* and *UB* report the lower and upper bound on the optimal solution achieved by the corresponding solving method of that model. Column *Gap* reports the optimality gap computed as $\frac{UB-LB}{UB}$. Column *Time* reports the solution time in seconds, and is only reported for the instances that are solved optimally within the time limit. In the tables, bold numbers indicate that a new best-known lower/upper bound is established.

We should note that the SOPLIB instances with a very small $\rho(R)$ have an optimal solution with $w_j = 0$ for all $j \in V$. This means that the optimal solution of these instance can be found much more efficiently by dropping the set of constraints (4.27)-(4.31) or their equivalent constraints from the other models. The CP solver is able to find the optimal solution faster on average compared to the MILP solver. However, neither solvers is able to inference the implied bounds on w_j.

Table 4.11: PCMCA-WT computational results of solving the models *CPB* and *RPB* with the MILP solver, and solving the models *CCP* and *RCP* with the CP Solver for TSPLIB instances.

				MILP Solver								CP Solver							
Instance				CPB				RPB				CCP				RCP			
Name	Size	$\rho(R)$	Best-Known	LB	UB	Gap	Time [s]	LB	UB	Gap	Time [s]	LB	UB	Gap	Time [s]	LB	UB	Gap	Time [s]
br17.12	18	0.359	[38, 44]	41	44	6.818	-	41	44	6.818	-	**44**	**44**	0.000	22.604	**44**	**44**	0.000	44.550
ESC07	9	0.611	1906	1906	1906	0.000	0.028	1906	1906	0.000	0.070	1906	1906	0.000	0.023	1906	1906	0.000	0.025
ESC11	13	0.359	21274	2174	2174	0.000	0.125	2174	2174	0.000	0.114	2174	2174	0.000	0.107	2174	2174	0.000	0.077
ESC12	14	0.396	1138	1138	1138	0.000	0.035	1138	1138	0.000	0.030	1138	1138	0.000	0.034	1138	1138	0.000	0.037
ESC25	27	0.177	1158	1158	1158	0.000	6.185	1158	1158	0.000	1.945	1158	1158	0.000	0.910	1158	1158	0.000	0.833
ESC47	49	0.108	747	747	747	0.000	59.760	747	747	0.000	22.153	747	747	0.000	3.886	747	747	0.000	2.708
ESC63	65	0.173	56	56	56	0.000	24.600	56	56	0.000	57.347	56	56	0.000	1.517	56	56	0.000	2.465
ESC78	80	0.139	1196	1196	1196	0.000	2410.483	1196	1196	0.000	257.609	1196	1196	0.000	100.511	1196	1196	0.000	18.971
ft53.1	54	0.082	4089	4089	4089	0.000	1764.235	4089	4089	0.000	2023.553	4089	4089	0.000	215.480	4089	4089	0.000	291.099
ft53.2	54	0.094	[4135, 4284]	4112	4317	4.749	-	**4161**	4334	3.992	-	4102	4284	4.248	-	4103	4284	4.225	-
ft53.3	54	0.225	[4729, 5359]	4746	5425	12.516	-	**4799**	**5279**	9.093	-	4493	5358	16.144	-	4508	5484	17.797	-
ft53.4	54	0.604	[5835, 6420]	5922	6420	7.757	-	**5923**	6420	7.741	-	5338	6502	17.902	-	5357	6420	16.558	-
ft70.1	71	0.036	[33128, 33298]	32777	33308	1.594	-	32827	33308	1.444	-	32669	33472	2.399	-	33101	33298	0.592	-
ft70.2	71	0.075	[33357, 33690]	33057	33977	2.708	-	33089	33916	2.438	-	32938	**33670**	2.174	-	32897	**33670**	2.296	-
ft70.3	71	0.142	[34298, 37974]	34152	38546	11.399	-	**34423**	38351	10.242	-	33825	36939	8.430	-	33813	**36932**	8.445	-
ft70.4	71	0.589	[36538, 40053]	36737	39145	6.151	-	**36850**	38771	4.955	-	33825	**36939**	8.430	-	35664	39843	10.489	-
rbg048a	50	0.444	[261, 263]	260	265	1.887	-	259	264	1.894	-	**263**	**263**	0.000	9.442	**263**	**263**	0.000	25.294
rbg050c	52	0.459	225	225	225	0.000	863.662	225	225	0.000	36.673	225	225	0.000	2.575	225	225	0.000	1.234
rbg109	111	0.909	[362, 414]	354	426	16.901	-	**366**	407	10.074	-	357	488	26.844	-	359	**401**	10.474	-
rbg150a	152	0.927	[456, 518]	447	511	12.524	-	461	**509**	9.430	-	**463**	591	21.658	-	461	517	10.832	-
rbg174a	176	0.929	[461, 580]	452	601	24.792	-	**463**	**553**	16.275	-	457	571	19.965	-	461	572	19.406	-
rbg253a	255	0.948	[529, 773]	523	1252	58.227	-	**532**	**718**	25.905	-	-	-	-	-	527	722	27.008	-
rbg323a	325	0.928	[969, 4035]	981	10111	90.298	-	974	2466	60.503	-	-	-	-	-	**1009**	**1891**	46.642	-
rbg341a	343	0.937	[758, 3800]	764	9313	91.796	-	761	2907	73.822	-	-	-	-	-	**780**	**1457**	46.465	-
rbg358a	360	0.886	[717, 3296]	**950**	11528	91.759	-	755	2453	69.221	-	-	-	-	-	788	**1150**	31.478	-
rbg378a	380	0.894	[649, 2759]	672	10242	93.439	-	648	2191	70.424	-	-	-	-	-	**678**	**1126**	39.787	-
kro124p.1	101	0.046	[32858, 35231]	32651	37120	12.039	-	32630	36099	9.610	-	32504	34100	4.680	-	32561	**33962**	4.125	-
kro124p.2	101	0.053	[33190, 37956]	32886	42573	22.754	-	33006	39931	17.342	-	32764	37074	11.625	-	32799	**35860**	8.536	-
kro124p.3	101	0.092	[34217, 51084]	33813	54183	37.595	-	34005	46764	27.284	-	33561	43910	23.569	-	33488	**42416**	21.049	-
kro124p.4	101	0.496	[39413, 55187]	**39969**	58944	32.192	-	39333	53456	26.420	-	38433	50910	24.508	-	37676	**49590**	24.025	-
p43.1	44	0.101	[2827, 4100]	2660	4085	34.884	-	2656	**3955**	32.845	-	**2860**	3955	27.686	-	2851	3990	28.546	-
p43.2	44	0.126	[2826, 4135]	991	4450	77.730	-	2705	4210	35.748	-	2856	4160	31.346	-	**2870**	4180	31.340	-
p43.3	44	0.191	[2864, 4880]	1067	5015	78.724	-	1383	4440	68.851	-	**2966**	4450	33.348	-	2897	**4255**	31.915	-
p43.4	44	0.164	[3101, 4620]	2995	5035	40.516	-	**3125**	4605	32.139	-	3090	**4495**	31.257	-	3094	4620	33.030	-
prob.100	100	0.048	[674, 1008]	668	2125	68.565	-	**677**	741	8.637	-	666	784	15.051	-	667	**738**	9.621	-
prob.42	42	0.116	171	171	171	0.000	396.458	171	171	0.000	230.506	171	171	0.000	79.667	171	171	0.000	34.245
ry48p.1	49	0.091	[13371, 13722]	13114	14272	8.114	-	13200	13670	3.438	-	13036	**13670**	4.638	-	13061	**13670**	4.455	-
ry48p.2	49	0.103	[13508, 14491]	13299	14415	7.742	-	13336	**14305**	6.774	-	13216	**14305**	7.613	-	13185	**14305**	7.829	-
ry48p.3	49	0.193	[14371, 15808]	13882	16193	14.272	-	13994	15840	11.654	-	13764	15546	11.463	-	13728	**15477**	11.301	-
ry48p.4	49	0.588	[17339, 19418]	17162	19744	13.077	-	17180	19583	12.271	-	16550	19837	16.570	-	16483	19495	15.450	-
Average						23.988	552.557			16.741	263.000			10.321	37.184			12.774	36.782

Table 4.12: PCMCA-WT computational results of solving the models *CPB* and *RPB* with the MILP solver, and solving the models *CCP* and *RCP* with the CP Solver for SOPLIB instances.

				MILP Solver								CP Solver							
Instance				CPB				RPB				CCP				RCP			
Name	Size	$\rho(R)$	Best-Known	LB	UB	Gap	Time [s]	LB	UB	Gap	Time [s]	LB	UB	Gap	Time [s]	LB	UB	Gap	Time [s]
R.200.100.1	200	0.020	29	29	29	0.000	18.394	29	29	0.000	6.017	29	29	0.000	28.271	29	29	0.000	31.403
R.200.100.15	200	0.847	[527, 1021]	497	1431	65.269	-	525	1033	49.177	-	381	1864	79.560	-	**589**	**979**	39.837	-
R.200.100.30	200	0.957	[787, 1898]	686	3252	78.905	-	774	**1761**	56.048	-	451	3001	84.972	-	**838**	1871	55.211	-
R.200.100.60	200	0.991	[8649, 18196]	8760	17004	48.483	-	**8861**	16930	47.661	-	6018	31561	80.932	-	8440	**16197**	47.892	
R.200.1000.1	200	0.020	887	887	887	0.000	1288.092	887	887	0.000	15.635	887	887	0.000	649.979	887	887	0.000	26.0915
R.200.1000.15	200	0.876	[6975, 16496]	6769	16336	58.564	-	6895	**12601**	45.282	-	5318	25196	78.893	-	**7231**	12812	43.561	-
R.200.1000.30	200	0.958	[10369, 24197]	9937	23226	57.216	-	**10512**	**22781**	53.856	-	7381	38410	80.784	-	10120	23249	56.471	-
R.200.1000.60	200	0.989	[11612, 21873]	11399	21706	47.485	-	**12042**	21993	45.246	-	6666	28522	76.629	-	10665	**19934**	46.498	
R.300.100.1	300	0.013	13	13	13	0.000	37.352	13	13	0.000	35.012	13	13	0.000	205.731	13	13	0.000	56.4263
R.300.100.15	300	0.905	[661, 3539]	660	6958	90.515	-	669	2259	70.385	-	-	-	-	-	**811**	**2056**	60.554	-
R.300.100.30	300	0.970	[1075, 3767]	1008	6790	85.155	-	1102	3163	65.160	-	-	-	-	-	**1157**	**2590**	55.328	-
R.300.100.60	300	0.994	[919, 2457]	919	4732	80.579	-	949	1954	51.433	-	-	-	-	-	**991**	**1865**	46.863	-
R.300.1000.1	300	0.013	715	715	715	0.000	3187.049	715	715	0.000	64.683	715	715	0.000	257.074	715	715	0.000	71.6789
R.300.1000.15	300	0.905	[7597, 23528]	7607	110366	93.107	-	7832	24047	67.430	-	-	-	-	-	**8768**	29423	70.200	-
R.300.1000.30	300	0.965	[11457, 40457]	11179	53835	79.235	-	12071	40863	70.460	-	-	-	-	-	**12269**	**31618**	61.196	-
R.300.1000.60	300	0.994	[10061, 30655]	10180	38212	73.359	-	10275	25323	59.424	-	-	-	-	-	**10408**	**21623**	51.866	-
R.400.100.1	400	0.010	6	6	376	98.404	-	6	6	0.000	995.137	6	6	0.000	726.057	6	6	0.000	97.3851
R.400.100.15	400	0.927	[836, 8807]	781	35044	97.771	-	856	22767	96.240	-	-	-	-	-	**963**	**3591**	73.183	-
R.400.100.30	400	0.978	[976, 7243]	911	39022	97.665	-	1010	26438	96.180	-	-	-	-	-	**1084**	**3061**	64.587	-
R.400.100.60	400	0.996	[847, 5545]	837	3309	74.705	-	861	2652	67.534	-	-	-	-	-	**966**	**2069**	53.311	-
R.400.1000.1	400	0.010	780	780	780	0.000	161.021	780	780	0.000	124.990	780	780	0.000	208.525	780	780	0.000	90.9555
R.400.1000.15	400	0.930	[8545, 85878]	8357	85878	90.269	-	9083	85878	89.423	-	-	-	-	-	**9976**	**35160**	71.627	-
R.400.1000.30	400	0.977	[11910, 95523]	11030	127290	91.335	-	11783	127290	90.743	-	-	-	-	-	**12337**	**57272**	78.459	-
R.400.1000.60	400	0.995	[10199, 55950]	9360	65615	85.735	-	9877	36662	73.059	-	-	-	-	-	9954	**22376**	55.515	-
R.500.100.1	500	0.008	3	3	3	0.000	2157.743	3	3	0.000	1881.297	3	3	0.000	2333.235	3	3	0.000	112.086
R.500.100.15	500	0.945	[1011, 11452]	964	11452	91.582	-	1018	11452	91.111	-	-	-	-	-	**1250**	**5508**	77.306	-
R.500.100.30	500	0.980	[980, 12225]	849	16963	94.995	-	976	14273	93.162	-	-	-	-	-	**1099**	**4841**	77.298	-
R.500.100.60	500	0.996	[811, 8427]	840	49105	98.289	-	840	6357	86.786	-	-	-	-	-	**931**	**2723**	65.810	-
R.500.1000.1	500	0.008	297	297	297	0.000	97.473	297	297	0.000	85.459	297	297	0.000	85.281	297	297	0.000	77.4382
R.500.1000.15	500	0.940	[8831, 107776]	8949	107776	91.697	-	9461	107776	91.222	-	-	-	-	-	**10628**	**45356**	76.568	
R.500.1000.30	500	0.981	[12419, 156359]	11799	156359	92.454	-	**12694**	156359	91.882	-	-	-	-	-	12576	**57330**	78.064	
R.500.1000.60	500	0.996	[8770, 33260]	8233	112466	92.680	-	8192	45696	82.073	-	-	-	-	-	6559	**20465**	67.950	-
R.600.100.1	600	0.007	[1, 379]	1	55	98.182	-	1	55	98.182	-	**1**	**1**	0.000	2710.470	**1**	**1**	0.000	2182.18
R.600.100.15	600	0.950	[831, 5949]	714	5931	87.962	-	845	4044	79.105	-	-	-	-	-	**938**	**2443**	61.605	-
R.600.100.30	600	0.985	[1106, 12875]	945	18932	95.008	-	1099	18932	94.195	-	-	-	-	-	740	**6467**	88.557	-
R.600.100.60	600	0.997	[805, 7893]	**838**	26732	96.865	-	778	25214	96.914	-	-	-	-	-	538	**2494**	78.428	-
R.600.1000.1	600	0.007	322	322	322	0.000	352.202	322	322	0.000	140.645	322	322	0.000	127.397	322	322	0.000	103.378
R.600.1000.15	600	0.945	[10600, 121877]	10753	121877	91.177	-	**10915**	121877	91.044	-	-	-	-	-	9401	**65039**	85.546	-
R.600.1000.30	600	0.984	[11668, 151010]	11352	190145	94.030	-	**12431**	190145	93.462	-	-	-	-	-	9356	**48775**	80.818	-
R.600.1000.60	600	0.997	[8027, 87770]	7962	256464	96.895	-	**8162**	75269	89.156	-	-	-	-	-	6908	**42652**	83.804	-
R.700.100.1	700	0.006	2	-	-	-	-	-	-	-	-	2	2	0.000	1649.486	2	2	0.000	619.22
R.700.100.15	700	0.957	[962, 6561]	815	14478	94.371	-	**972**	5718	83.001	-	-	-	-	-	655	**2759**	76.260	-
R.700.100.30	700	0.987	[843, 20281]	896	6960	87.126	-	**983**	4218	76.695	-	-	-	-	-	588	**2531**	76.768	-
R.700.100.60	700	0.997	[567, 7032]	538	7033	92.350	-	555	1854	70.065	-	-	-	-	-	383	**1598**	76.033	-
R.700.1000.1	700	0.006	611	611	616	0.812	-	611	616	0.812	-	611	611	0.000	592.107	611	611	0.000	368.139
R.700.1000.15	700	0.956	[5075, 10779]	4375	147321	97.030	-	**5136**	7145	28.118	-	-	-	-	-	2787	**6315**	55.867	-
R.700.1000.30	700	0.986	[4777, 9829]	4477	32742	86.326	-	**4827**	6981	30.855	-	-	-	-	-	2658	**6115**	56.533	-
R.700.1000.60	700	0.997	[3004, 15579]	2942	8534	65.526	-	2997	5842	48.699	-	-	-	-	-	1913	**5357**	64.290	-
Average						68.917	912.416			57.687	372.097			26.765	797.801			49.160	319.698

Table 4.13: PCMCA-WT computational results of solving the models *CPB* and *RPB* with the MILP solver, and solving the models *CCP* and *RCP* with the CP Solver for COMPILERS instances.

Instance				MILP Solver								CP Solver							
				CPB				RPB				CCP				RCP			
Name	Size	$\rho(R)$	Best-Known	LB	UB	Gap	Time [s]	LB	UB	Gap	Time [s]	LB	UB	Gap	Time [s]	LB	UB	Gap	Time [s]
gsm.153.124	126	0.970	[256, 311]	257	312	17.628	-	269	311	13.505	-	278	317	12.303	-	**280**	311	9.968	-
gsm.444.350	353	0.990	[2406, 2873]	2294	4878	52.973	-	2405	4856	50.474	-	-	-	-	-	**2456**	4310	43.016	-
gsm.462.77	79	0.840	[408, 468]	402	478	15.900	-	402	477	15.723	-	**419**	474	11.603	-	418	**465**	10.108	-
jpeg.1483.25	27	0.484	87	87	87	0.000	26.041	87	87	0.000	18.556	87	87	0.000	1.194	87	87	0.000	1.071
jpeg.3184.107	109	0.887	[508, 665]	506	**656**	22.866	-	510	715	28.671	-	**518**	718	27.855	-	517	692	25.289	-
jpeg.3195.85	87	0.740	[22, 25]	17	25	32.000	-	17	25	32.000	-	**23**	25	8.000	-	22	25	12.000	-
jpeg.3198.93	95	0.752	[180, 204]	180	**188**	4.255	-	180	**188**	4.255	-	**181**	**188**	3.723	-	**181**	**188**	3.723	-
jpeg.3203.135	137	0.897	[616, 750]	602	980	38.571	-	618	751	17.710	-	**629**	913	31.106	-	626	**750**	16.533	-
jpeg.3740.15	17	0.257	33	33	33	0.000	1.523	33	33	0.000	0.839	33	33	0.000	0.157	33	33	0.000	0.095
jpeg.4154.36	38	0.633	90	90	90	0.000	556.798	90	90	0.000	60.924	90	90	0.000	1.272	90	90	0.000	1.764
jpeg.4753.54	56	0.769	164	164	164	0.000	2753.752	164	164	0.000	1790.269	164	164	0.000	15.342	164	164	0.000	16.877
susan.248.197	199	0.939	[804, 1184]	792	1978	59.960	-	802	1370	41.460	-	**805**	1361	40.852	-	780	1320	40.909	-
susan.260.158	160	0.916	[576, 876]	568	937	39.381	-	573	938	38.913	-	596	991	39.859	-	**598**	897	33.333	-
susan.343.182	184	0.936	[621, 830]	617	798	22.682	-	622	**776**	19.845	-	**636**	1043	39.022	-	632	792	20.202	-
typeset.10192.123	125	0.744	[280, 415]	274	429	36.131	-	282	**379**	25.594	-	**293**	385	23.896	-	292	387	24.548	-
typeset.10835.26	28	0.349	[99, 111]	99	111	10.811	-	100	112	10.714	-	**110**	111	0.901	-	109	111	1.802	-
typeset.12395.43	45	0.518	[143, 146]	140	146	4.110	-	141	146	3.425	-	**146**	**146**	0.000	2181.942	**146**	**146**	0.000	2780.121
typeset.15087.23	25	0.557	97	97	97	0.000	60.502	97	97	0.000	29.118	97	97	0.000	0.477	97	97	0.000	0.318
typeset.15577.36	38	0.555	125	125	125	0.000	286.210	125	125	0.000	43.164	125	125	0.000	2.116	125	125	0.000	1.713
typeset.16000.68	70	0.658	[77, 80]	66	81	18.519	-	66	80	17.500	-	**79**	80	1.250	-	71	80	11.250	-
typeset.1723.25	27	0.245	60	60	60	0.000	590.577	60	60	0.000	86.068	60	60	0.000	4.013	60	60	0.000	3.469
typeset.19972.246	248	0.993	[1427, 1929]	1422	3562	60.079	-	1452	2509	42.128	-	1519	2961	48.700	-	**1525**	2804	45.613	-
typeset.4391.240	242	0.981	[1132, 1412]	1108	2595	57.303	-	1137	2476	54.079	-	1149	2511	54.241	-	**1154**	1905	39.423	-
typeset.4597.45	47	0.493	[150, 155]	150	154	2.597	-	151	154	1.948	-	**154**	**154**	0.000	209.659	**154**	**154**	0.000	128.916
typeset.4724.433	435	0.995	[2576, 3433]	-	-	-	-	2673	6131	56.402	-	-	-	-	-	**2679**	7194	62.761	-
typeset.5797.33	35	0.748	113	113	113	0.000	851.490	113	113	0.000	28.504	113	113	0.000	0.547	113	113	0.000	0.574
typeset.5881.246	248	0.986	[1411, 1700]	1378	2258	38.973	-	1396	2426	42.457	-	1406	2385	41.048	-	1394	2084	33.109	-
Average						20.567	640.862			19.141	257.180			15.374	241.672			16.059	293.492

4.7 Conclusions

In this chapter we introduced an extension on the *Precedence-Constrained Minimum-Cost Arborescence* problem, named the *Precedence-Constrained Minimum-Cost Arborescence with Waiting-Times*, and presented a proof that the problem belongs to the $\mathcal{NP}$-hard complexity class through a reduction to the *Rectilinear Steiner Arborescence problem*.

Several MILP models for the PCMCA-WT are devised, by extending and modi-

fying some of the MILP models proposed for the PCMCA problem. The Computational results has shown that for a total of 68 instances, the model *DA* achieves the lowest average optimality gap of 13.8%, followed by the model *RPB* with an average optimality gap of 15.5%, followed the model *AAC* with an average optimality gap of 15.7%. Furthermore, for a total of 11 out of the same 68 instances, the model *RPB* is 73.9% faster on average at finding the optimal solution for those instances compared to the model *DA*, and 43.1% faster on average compared to the model *AAC*. Finally, for a total of 116 instances, the model *RPB* is able to find the best lower and upper bound for a larger number of instances (68 out of 116 instances) compared to the models *DA* (37 out of 116 instances) and *AAC* (26 out of 116 instances). Considering all factors, we can conclude that the model *RPB* has the best overall performance on the benchmark instances considered.

Finally, we proposed two polynomial sized CP models that are based on the two models *CPB* and *RBP*. The computational experiments conducted have shown that increasing the number of threads used by the MILP Solver considerably improves the performance of the two MILP models *CPB* and *RBP*. In addition, the computational experiments have shown that the CP models are often able to find better quality solutions compared to the two MILP models, however the proposed MILP models are often better at finding tighter lower bound estimates on the value of the optimal solution. Finally, by utilizing a parallel MILP solver and CP solver, we were able to find the optimal solution to six additional instances, and improved lower bounds for 70 instances, and improved upper bounds for 71 instances, out of a total of 87 open instances. The best-known bounds for the benchmark instances (116 instances) can be found in tables 4.14-4.16, that contain a total of 80 open instances.

Table 4.14: PCMCA-WT best-known solutions for TSPLIB instances.

Name	Size	$\rho(R)$	Best-Known Solution
br17.10	18	0.314	44
br17.12	18	0.359	44
ESC07	9	0.611	1906
ESC11	13	0.359	2174
ESC12	14	0.396	1138
ESC25	27	0.177	1158
ESC47	49	0.108	747
ESC63	65	0.173	56
ESC78	80	0.139	1196
ft53.1	54	0.082	4089
ft53.2	54	0.094	[4161, 4284]
ft53.3	54	0.225	[4799, 5279]
ft53.4	54	0.604	[5923, 6420]
ft70.1	71	0.036	[33101, 33298]
ft70.2	71	0.075	[33089, 33670]
ft70.3	71	0.142	[34423, 36932]
ft70.4	71	0.589	[36850, 36939]
rbg048a	50	0.444	263
rbg050c	52	0.459	225
rbg109	111	0.909	[366, 401]
rbg150a	152	0.927	[463, 509]
rbg174a	176	0.929	[463, 553]
rbg253a	255	0.948	[532, 718]
rbg323a	325	0.928	[1009, 1891]
rbg341a	343	0.937	[780, 1457]
rbg358a	360	0.886	[950, 1150]
rbg378a	380	0.894	[678, 1126]
kro124p.1	101	0.046	[32651, 33962]
kro124p.2	101	0.053	[33006, 35860]
kro124p.3	101	0.092	[34005, 42416]
kro124p.4	101	0.496	[39969, 49590]
p43.1	44	0.101	[2860, 3955]
p43.2	44	0.126	[2870, 4160]
p43.3	44	0.191	[2966, 4255]
p43.4	44	0.164	[3125, 4495]
prob.100	100	0.048	[677, 738]
prob.42	42	0.116	171
ry48p.1	49	0.091	[13200, 13670]
ry48p.2	49	0.103	[13336, 14305]
ry48p.3	49	0.193	[13994, 15477]
ry48p.4	49	0.588	[17180, 19495]

Table 4.15: PCMCA-WT best-known solutions for SOPLIB instances.

Name	Size	$\rho(R)$	Best-Known Solution
R.200.100.1	200	0.020	29
R.200.100.15	200	0.847	[589, 979]
R.200.100.30	200	0.957	[838, 1761]
R.200.100.60	200	0.991	[8861, 16197]
R.200.1000.1	200	0.020	887
R.200.1000.15	200	0.876	[7231, 12601]
R.200.1000.30	200	0.958	[10512, 22781]
R.200.1000.60	200	0.989	[12042, 19934]
R.300.100.1	300	0.013	13
R.300.100.15	300	0.905	[811, 2056]
R.300.100.30	300	0.970	[1157, 2590]
R.300.100.60	300	0.994	[991, 1865]
R.300.1000.1	300	0.013	715
R.300.1000.15	300	0.905	[8768, 24047]
R.300.1000.30	300	0.965	[12269, 31618]
R.300.1000.60	300	0.994	[10408, 21623]
R.400.100.1	400	0.010	6
R.400.100.15	400	0.927	[963, 3591]
R.400.100.30	400	0.978	[1084, 3061]
R.400.100.60	400	0.996	[966, 2069]
R.400.1000.1	400	0.010	780
R.400.1000.15	400	0.930	[9976, 35160]
R.400.1000.30	400	0.977	[12337, 57272]
R.400.1000.60	400	0.995	[9954, 22376]
R.500.100.1	500	0.008	3
R.500.100.15	500	0.945	[1250, 5508]
R.500.100.30	500	0.980	[1099, 4841]
R.500.100.60	500	0.996	[931, 2723]
R.500.1000.1	500	0.008	297
R.500.1000.15	500	0.940	[10628, 45356]
R.500.1000.30	500	0.981	[12694, 57330]
R.500.1000.60	500	0.996	[8233, 20465]
R.600.100.1	600	0.007	1
R.600.100.15	600	0.950	[938, 2443]
R.600.100.30	600	0.985	[1099, 6467]
R.600.100.60	600	0.997	[838, 2494]
R.600.1000.1	600	0.007	322
R.600.1000.15	600	0.945	[10915, 65039]
R.600.1000.30	600	0.984	[12431, 48775]
R.600.1000.60	600	0.997	[8162, 42652]
R.700.100.1	700	0.006	2
R.700.100.15	700	0.957	[972, 2759]
R.700.100.30	700	0.987	[983, 2531]
R.700.100.60	700	0.997	[555, 1598]
R.700.1000.1	700	0.006	611
R.700.1000.15	700	0.956	[5136, 6315]
R.700.1000.30	700	0.986	[4827, 6115]
R.700.1000.60	700	0.997	[2997, 5357]

Table 4.16: PCMCA-WT best-known solutions for COMPILERS instances.

Name	Size	$\rho(R)$	Best-Known Solution
gsm.153.124	126	0.970	[280, 311]
gsm.444.350	353	0.990	[2456, 4310]
gsm.462.77	79	0.840	[419, 465]
jpeg.1483.25	27	0.484	87
jpeg.3184.107	109	0.887	[518, 656]
jpeg.3195.85	87	0.740	[23, 25]
jpeg.3198.93	95	0.752	188
jpeg.3203.135	137	0.897	[629, 750]
jpeg.3740.15	17	0.257	33
jpeg.4154.36	38	0.633	90
jpeg.4753.54	56	0.769	164
susan.248.197	199	0.939	[805, 1320]
susan.260.158	160	0.916	[598, 897]
susan.343.182	184	0.936	[636, 776]
typeset.10192.123	125	0.744	[293, 379]
typeset.10835.26	28	0.349	[110, 111]
typeset.12395.43	45	0.518	146
typeset.15087.23	25	0.557	97
typeset.15577.36	38	0.555	125
typeset.16000.68	70	0.658	[79, 80]
typeset.1723.25	27	0.245	60
typeset.19972.246	248	0.993	[1525, 2509]
typeset.4391.240	242	0.981	[1154, 1905]
typeset.4597.45	47	0.493	154
typeset.4724.433	435	0.995	[2679, 6131]
typeset.5797.33	35	0.748	113
typeset.5881.246	248	0.986	[1406, 2084]

Chapter 5

Conclusions

In this thesis, we have introduced and studied the *Precedence-Constrained Minimum-Cost Arborescence Problem*, and its extension, the *Precedence-Constrained Minimum-Cost Arborescence Problem with Waiting-Times*. For both problems, we presented proofs which shows that both problems belong to the $\mathcal{NP}$-hard complexity class, therefore we utilized several techniques from the area of combinatorial optimizations for solving the problems. A computational study for the different techniques used is presented, with the aim of studying the performance of each method in terms of solution time, memory consumption, and solution quality.

The first part of the thesis introduced several MILP models for the PCMCA, by extending selected models previously proposed in the literature for the MCA problem. The models are extended by adding precedence-enforcing constraints to models in order to satisfy the precedence relationships between vertex pairs. The first precedence-enforcing constraints proposed are based on modifying the flow-conservation constraints of a Multicommodity flow model. Next, we introduced precedence-enforcing constraints that eliminate precedence-violating paths through value propagation along the paths of a potential solution. Finally, we extended the classical connectivity cover constraint of the MCA in order to satisfy the prece-

dence relationships between vertex pairs. The computational study conducted has shown that the latter model has the best overall performance compared to the other proposed models. Finally, we have introduced a B&B algorithm for the PCMCA problem that is based on a Lagragian relaxation of the best performing model for the PCMCA. The B&B algorithm was evaluated using several step-size rules in order to solve the Lagrangian relaxation. The computational study conducted has shown that the B&B algorithm is able to find the optimal solution faster when compared to solving the same model with an MILP solver, except on a very small subset of instances.

The second part of this thesis introduced several MILP models for the PCMCA-WT, by extending a subset of the models introduced for the PCMCA. Furthermore, we introduced a new polynomial sized model for the PCMCA-WT, and by exploiting a special feature of the problem we were able to further reduced the number of variables and constraints in the model. The computational study conducted has shown that the polynomial sized model with the reduced number of variables and constraints has the best overall performance compared to the other models proposed for the PCMCA-WT. Compared to the PCMCA, the models perform less effectively on PCMCA-WT instances, and therefore we have considered using a parallel MILP Solver and a parallel CP Solver to enhance the performance of the model and improve the solution quality of the benchmark instances. The performance of the MILP and CP Solvers was evaluated by solving the polynomial sized models, due to limitations of the utilized CP solver, and based on the fact that the *Reduced Path-Based model* has the best overall performance at solving the PCMCA-WT. The computational study conducted has shown that the CP Solver has an overall better performance compared to the MILP Solver on the considered model. We should note that, utilizing the CP Solver in order to solve the models proposed for the PCMCA was not considered, as the MILP Solver performs sufficiently well on the considered

benchmark instances.

Future work on the PCMCA could consider using more effective techniques for solving the Lagrangian relaxation, and faster converging step-size rules compared to the ones considered in this study. Furthermore, more effective pruning techniques could be considered in order to reduce the size of the solution space explored by the B&B algorithm. Finally, the performance of the models should be evaluated on larger size instances with different structures and size of precedence relationships. For the PCMCA-WT, future work could investigate finding new valid inequalities for the problem, combining MILP and CP solvers in order to utilize their different strengths, and possibly extending the B&B algorithm proposed for the PCMCA to solve instances of the PCMCA-WT.

Bibliography

[1] Aarts, E., Aarts, E. H. L., & Lenstra, J. K. (2003). *Local search in combinatorial optimization*. Princeton University Press.

[2] Abdelmaguid, T. F. (2018). An efficient mixed integer linear programming model for the minimum spanning tree problem. *Mathematics*, *6*(10), 183.

[3] Ascheuer, N., Jünger, M., & Reinelt, G. (2000). A branch & cut algorithm for the asymmetric traveling salesman problem with precedence constraints. *Computational Optimization and Applications*, *17*(1), 61–84.

[4] Bang-Jensen, J. (1991). Edge-disjoint in-and out-branchings in tournaments and related path problems. *Journal of Combinatorial Theory, Series B*, *51*(1), 1–23.

[5] Bazaraa, M. S., & Sherali, H. D. (1981). On the choice of step size in subgradient optimization. *European Journal of Operational Research*, *7*(4), 380–388.

[6] Bérczi, K., Fujishige, S., & Kamiyama, N. (2009). A linear-time algorithm to find a pair of arc-disjoint spanning in-arborescence and out-arborescence in a directed acyclic graph. *Information processing letters*, *109*(23-24), 1227–1231.

[7] Bock, F. (1971). An algorithm to construct a minimum directed spanning tree in a directed network. *Developments in operations research*, (pp. 29–44).

[8] Böcker, S., & Rasche, F. (2008). Towards de novo identification of metabolites by analyzing tandem mass spectra. *Bioinformatics*, *24*(16), i49–i55.

[9] Cai, M., Deng, X., & Wang, L. (2004). Minimum k arborescences with bandwidth constraints. *Algorithmica*, *38*(4), 529–537.

[10] Carrabs, F., Cerulli, R., Pentangelo, R., & Raiconi, A. (2021). Minimum spanning tree with conflicting edge pairs: a branch-and-cut approach. *Annals of Operations Research*, *298*(1), 65–78.

[11] Chekuri, C., Khanna, S., & Shepherd, F. B. (2005). Multicommodity flow, well-linked terminals, and routing problems. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, (pp. 183–192).

[12] Chou, X., Dell'Amico, M., Jamal, J., & Montemanni, R. (2023). Precedence-constrained arborescences. *European Journal of Operational Research*, *307*(2), 575–589.

[13] Chu, Y.-J. (1965). On the shortest arborescence of a directed graph. *Scientia Sinica*, *14*, 1396–1400.

[14] Cococcioni, M., & Fiaschi, L. (2021). The big-m method with the numerical infinite m. *Optimization Letters*, *15*(7), 2455–2468.

[15] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). *Introduction to algorithms*. MIT press.

[16] Dantzig, G. B. (1990). Origins of the simplex method. In *A history of scientific computing*, (pp. 141–151). Springer Science & Business Media.

[17] Darmann, A., Pferschy, U., & Schauer, J. (2009). Determining a minimum spanning tree with disjunctive constraints. In *International Conference on Algorithmic Decision Theory*, (pp. 414–423). Springer.

[18] Dasgupta, D., & Michalewicz, Z. (2013). *Evolutionary algorithms in engineering applications*. Springer Science & Business Media.

[19] Dell'Amico, M., Jamal, J., & Montemanni, R. (2021). A mixed integer linear program for a precedence-constrained minimum-cost arborescence problem. In *2021 The 8th International Conference on Industrial Engineering and Applications (Europe)*, (pp. 216–221).

[20] Dell'Amico, M., Jamal, J., & Montemanni, R. (2023). A branch-and-bound algorithm for the precedence-constrained minimum-cost arborescence problem. *Computers & Operations Research*, (p. 106248).

[21] Dell'Amico, M., Jamal, J., & Montemanni, R. (2023). Compact models for the precedence-constrained minimum-cost arborescence problem. In *Advances in Intelligent Traffic and Transportation Systems*, (pp. 112–126). IOS Press.

[22] Dell'Amico, M., Jamal, J., & Montemanni, R. (2023). Compact models for the precedence-constrained minimum-cost arborescence problem with waiting-times. *Annals of Operations Research (under review)*.

[23] Derboni, M., Rizzoli, A. E., Montemanni, R., Jamal, J., Kovacs, N., & Cellina, F. (2018). Challenges and opportunities in deploying a mo-bility platform integrating public transport and car-pooling services. In *Proceedings of the 18th Swiss Transport Research Conference*.

[24] Dowsland, K. A., & Thompson, J. (2012). Simulated annealing. *Handbook of natural computing*, (pp. 1623–1655).

[25] Duhamel, C., Gouveia, L., Moura, P., & Souza, M. (2008). Models and heuristics for a minimum arborescence problem. *Networks: An International Journal*, *51*(1), 34–47.

[26] Edmonds, J. (1967). Optimum branchings. *Journal of Research of the national Bureau of Standards B*, *71*(4), 233–240.

[27] Escudero, L. F. (1988). An inexact algorithm for the sequential ordering problem. *European Journal of Operational Research*, *37*(2), 236–249.

[28] Escudero, L. F., Guignard, M., & Malik, K. (1994). A lagrangian relax-and-cut approach for the sequential ordering problem with precedence relationships. *Annals of Operations Research*, *50*(1), 219–237.

[29] Eswaran, K. P., & Tarjan, R. E. (1976). Augmentation problems. *SIAM Journal on Computing*, *5*(4), 653–665.

[30] Fertin, G., Fradin, J., & Jean, G. (2017). Algorithmic aspects of the maximum colorful arborescence problem. In *International Conference on Theory and Applications of Models of Computation*, (pp. 216–230). Springer.

[31] Fischetti, M., & Toth, P. (1993). An efficient algorithm for the min-sum arborescence problem on complete digraphs. *ORSA Journal on Computing*, *5*(4), 426–434.

[32] Fischetti, M., & Vigo, D. (1997). A branch-and-cut algorithm for the resource-constrained minimum-weight arborescence problem. *Networks: An International Journal*, *29*(1), 55–67.

[33] Fisher, M. L. (1981). The lagrangian relaxation method for solving integer programming problems. *Management science*, *27*(1), 1–18.

[34] Ford, L. R., & Fulkerson, D. R. (1956). Maximal flow through a network. *Canadian journal of Mathematics*, *8*, 399–404.

[35] Fredman, M. L., & Tarjan, R. E. (1987). Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, *34*(3), 596–615.

[36] Frieze, A. M., & Tkocz, T. (2022). A randomly weighted minimum arborescence with a random cost constraint. *Mathematics of Operations Research*, *47*(2), 1664–1680.

[37] 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.

[38] Galbiati, G., Gualandi, S., & Maffioli, F. (2011). On minimum changeover cost arborescences. In *International Symposium on Experimental Algorithms*, (pp. 112–123). Springer.

[39] Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability: A Guide to the Theory of NP-Completeness*. W H Freeman & Co., San Francisco.

[40] Georgiadis, L. (2003). Arborescence optimization problems solvable by edmonds' algorithm. *Theoretical Computer Science*, *301*(1-3), 427–437.

[41] Glover, F., & Laguna, M. (1998). Tabu search. In *Handbook of combinatorial optimization*, (pp. 2093–2229). Springer.

[42] Glover, F. W., & Kochenberger, G. A. (2006). *Handbook of metaheuristics*, vol. 57. Springer Science & Business Media.

[43] Goemans, M. X. (2005). Lecture notes on the ellipsoid algorithm.

[44] Goemans, M. X. (2007). Lecture notes on the arborescence problem.

[45] Gogna, A., & Tayal, A. (2013). Metaheuristics: review and application. *Journal of Experimental & Theoretical Artificial Intelligence*, *25*(4), 503–526.

[46] Google (2015). Google OR-Tools. [last accessed 10-March-2023].
URL `https://developers.google.com/optimization`

[47] Gouveia, L., & Lopes, M. J. (2005). The capacitated minimum spanning tree problem: On improved multistar constraints. *European Journal of Operational Research*, *160*(1), 47–62.

[48] Grakova, E., Golasowski, M., Montemanni, R., Slaninová, K., Martinovič, J., Jamal, J., Janurová, K., & Salani, M. (2019). Hyperparameter search in periodic vehicle routing problem. In *MATEC Web of Conferences*, vol. 259, (p. 01003). EDP Sciences.

[49] Hakimi, S. L. (1965). Optimum distribution of switching centers in a communication network and some related graph theoretic problems. *Operations research*, *13*(3), 462–475.

[50] Hao, J., & Orlin, J. B. (1994). A faster algorithm for finding the minimum cut in a directed graph. *Journal of Algorithms*, *17*(3), 424–446.

[51] Henschel, R., Leal-Taixé, L., & Rosenhahn, B. (2014). Efficient multiple people tracking using minimum cost arborescences. In *German Conference on Pattern Recognition*, (pp. 265–276). Springer.

[52] Hoffman, A. J. (1974). A generalization of max flow—min cut. *Mathematical Programming*, *6*(1), 352–359.

[53] IBM (1988). IBM CPLEX Optimizer. [last accessed 10-March-2023]. URL `https://www.ibm.com/de-de/analytics/cplex-optimizer`

[54] Jamal, J., Loske, D., Klumpp, M., Chou, X., Di Florio Di Renzo, A., Dell'Amico, M., & Montemanni, R. (2022). Skill-based joint order batching and picker routing problem. In *2022 The 9th International Conference on Industrial Engineering and Applications (Europe)*, (pp. 64–69).

[55] Jamal, J., & Montemanni, R. (2018). Industrial cluster symbiosis optimisation based on linear programming. *Process Integration and Optimization for Sustainability*, *2*(4), 353–364.

[56] Jamal, J., Montemanni, R., Huber, D., Derboni, M., & Rizzoli, A. E. (2017). A multi-modal and multi-objective journey planner for integrating carpooling and public transport. *Journal of Traffic and Logistics Engineering Vol*, *5*(2), 68–72.

[57] Jamal, J., Rizzoli, A. E., Montemanni, R., & Huber, D. (2016). Tour planning and ride matching for an urban social carpooling service. In *MATEC Web of Conferences*, vol. 81, (p. 04010). EDP Sciences.

[58] Jamal, J., Shobaki, G., Papapanagiotou, V., Gambardella, L. M., & Montemanni, R. (2017). Solving the sequential ordering problem using branch and bound. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, (pp. 1–9). IEEE.

[59] Kawatra, R., & Bricker, D. (2004). Design of a degree-constrained minimal spanning tree with unreliable links and node outage costs. *European Journal of Operational Research*, *156*(1), 73–82.

[60] Kleinberg, J., & Tardos, E. (2006). *Algorithm design*. Pearson Education India.

[61] Kováč, J. (2013). Complexity of the path avoiding forbidden pairs problem revisited. *Discrete Applied Mathematics*, *161*(10-11), 1506–1512.

[62] Kramer, O. (2017). Genetic algorithms. In *Genetic algorithm essentials*, (pp. 11–19). Springer.

[63] 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)*, (pp. 1204–1209). IEEE.

[64] Li, Y., Thai, M. T., Wang, F., & Du, D.-Z. (2006). On the construction of a strongly connected broadcast arborescence with bounded transmission delay. *IEEE Transactions on mobile computing*, *5*(10), 1460–1470.

[65] Lucena, A. (1992). Steiner problem in graphs: Lagrangean relaxation and cutting planes. *Coal Bulletin*, *21*(2), 2–8.

[66] Manson, S. M. (2001). Simplifying complexity: a review of complexity theory. *Geoforum*, *32*(3), 405–414.

[67] Miller, C. E., Tucker, A. W., & Zemlin, R. A. (1960). Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, *7*(4), 326–329.

[68] Montemanni, R., & Jamal, J. (2018). Industrial cluster optimization based on linear programming. In *2018 5th International Conference on Industrial Engineering and Applications (ICIEA)*, (pp. 241–246). IEEE.

[69] Montemanni, R., Smith, D. H., & Gambardella, L. M. (2008). A heuristic manipulation technique for the sequential ordering problem. *Computers & Operations Research*, *35*(12), 3931–3944.

[70] Morais, V., Gendron, B., & Mateus, G. R. (2019). The p-arborescence star problem: Formulations and exact solution approaches. *Computers & Operations Research*, *102*, 91–101.

[71] Ognibene Pietri, N., Chou, X., Loske, D., Klumpp, M., Jamal, J., & Montemanni, R. (2022). The picking and packing problem in buy-online-pick-up-in-store retailing. In *Operations Research Proceedings 2021: Selected Papers of the International Conference of the Swiss, German and Austrian Operations Research Societies (SVOR/ASRO, GOR eV, ÖGOR), University of Bern, Switzerland, August 31–September 3, 2021*, (pp. 239–244). Springer.

[72] Papadimitriou, C. H., & Steiglitz, K. (1998). *Combinatorial optimization: algorithms and complexity*. Courier Corporation.

[73] Papapanagiotou, V., Jamal, J., Montemanni, R., Shobaki, G., & Gambardella, L. M. (2015). A comparison of two exact algorithms for the sequential ordering problem. In *2015 IEEE conference on systems, process and control (ICSPC)*, (pp. 73–78). IEEE.

[74] Pereira, A. H., Mateus, G. R., & Urrutia, S. (2022). Branch-and-cut algorithms for the-arborescence star problem. *International Transactions in Operational Research*, *29*(4), 2374–2400.

[75] Reinelt, G. (1991). TSPLIB-—A traveling salesman problem library. *ORSA journal on computing*, *3*(4), 376–384.

[76] Roos, C., Terlaky, T., & Vial, J. P. (2005). *Interior point methods for linear optimization*. Springer Science & Business Media.

[77] Schrijver, A. (1998). *Theory of linear and integer programming*. John Wiley & Sons.

[78] Shi, W., & Su, C. (2005). The rectilinear steiner arborescence problem is np-complete. *SIAM Journal on Computing*, *35*(3), 729–740.

[79] Shobaki, G., & Jamal, J. (2015). An exact algorithm for the sequential ordering problem and its application to switching energy minimization in compilers. *Computational Optimization and Applications*, *61*(2), 343–372.

[80] Shobaki, G., Rmaileh, N. E. A., & Jamal, J. (2016). Studying the impact of bit switching on cpu energy. In *Proceedings of the 19th International Workshop on software and compilers for embedded systems*, (pp. 173–179).

[81] S&P Global (2007). Belarussian president threatens new duties on russian oil transit. [Online; posted 29-January-2007; last accessed 10-March-2023]. URL `https://www.spglobal.com/marketintelligence/en/mi/country-industry-forecasting.html?ID=106598459`

[82] Talbi, E.-G. (2009). *Metaheuristics: from design to implementation*. John Wiley & Sons.

[83] Tarjan, R. (1972). Depth-first search and linear graph algorithms. *SIAM journal on computing*, *1*(2), 146–160.

[84] Toth, P., & Vigo, D. (1995). An exact algorithm for the capacitated shortest spanning arborescence. *Annals of Operations Research*, *61*(1), 121–141.

[85] Van Beek, P. (2006). Backtracking search algorithms. In *Foundations of artificial intelligence*, vol. 2, (pp. 85–134). Elsevier.

[86] Viana, L. A. d. C., & Campêlo, M. (2020). Two dependency constrained spanning tree problems. *International Transactions in Operational Research*, *27*(2), 867–898.

[87] Weiss, M. A. (1995). *Data structures and algorithm analysis*. Benjamin-Cummings Publishing Co., Inc.

[88] Wolsey, L. A., & Nemhauser, G. L. (1999). *Integer and combinatorial optimization*, vol. 55. John Wiley & Sons.