Proceedings of the 18th Conference on Computer Science and Intelligence Systems pp. 421–430
DOI: 10.15439/2023F6191
ISSN 2300-5963 ACSIS, Vol. 35

Modelling and Solving the Precedence-Constrained Minimum-Cost Arborescence Problem with Waiting-Times

Mauro Dell'Amico, Jafar Jamal, Roberto Montemanni
0000-0002-3283-6131
0009-0003-9368-1094
0000-0002-0229-0465
Department of Sciences and Methods for Engineering
University of Modena and Reggio Emilia
Via Amendola 2, 42122 Reggio Emilia, Italy
Email: mauro.dellamico@unimore.it;
268108@studenti.unimore.it;
roberto.montemanni@unimore.it

***Abstract*—A polynomial-size mixed integer linear programming model for the Precedence-Constrained Minimum-Cost Arborescence Problem with Waiting-Times was recently proposed in the literature, that uses a smaller number of variables and constraints compared to previously proposed polynomial-size models. In this work, we extend this model with constraint programming constructs to further enhance its performance. An extensive computational study support that modern constraint programming solvers are the best tool available at solving the models proposed. Several improvements to state-of-the-art results are finally reported.**

***Index Terms*—Combinatorial Optimization; Arborescences; Precedence-Constraints.**

I. Introduction

THE *Minimum-Cost Arborescence* (MCA) problem involves finding a directed minimum-cost spanning tree, rooted at vertex r, in a given input directed graph. Jack Edmonds [1], and Yoeng-Jin Chu and Tseng-Hong Liu [2] independently introduced the first polynomial time algorithm for solving the problem. Gabow and Tarjan [3] improved the running time of the algorithm by using *disjoint-sets* and a special implementation of *Fibonacci heaps*.

Several variations of the MCA problem with different objective function and/or constraints were introduced in the literature since its introduction. Given a finite resource associated with each vertex in the input graph, the *Resource-Constrained Minimum-Weight Arborescence problem* [4] is an $\mathcal{NP}$-hard problem which asks 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. Given an integer Q and non-negative integer vertex demand q_j associated with each vertex, the *Capacitated Minimum Spanning Tree problem* [5] is an $\mathcal{NP}$-hard problem which asks to find a directed minimum spanning tree rooted at r, such that the sum of the weights of the vertices in any subtree off the root is at most Q. Given a weighted directed acyclic graph with each vertex having a specified color from a set of colors, the *Maximum Colorful Arborescence problem* [6] is an $\mathcal{NP}$-hard problem which asks to find an arborescence of maximum weight, in which no color appears more than once. Given an integer rank associated with each vertex, the *Restricted Fathers Tree problem* [7] asks to find a minimum-cost arborescence rooted at r, such that the path between each vertex and the root contains only vertices with same rank or higher.

Constraint programming (CP) is paradigm for solving combinatorial problems by representing them as *constraint satisfaction problems* (CSP) [8]. A CSP is represented as a set of variables each with a defined domain of values, and a set of relations/constraints on the subsets of these variables. A CP solver takes a CSP and finds an assignment to all the variables that satisfies the constraints, and can also extend the problem to finding optimal solutions according to an optimization criteria. A CP solver searches the solution space systematically using a branch-and-bound algorithm with inference techniques which consists of propagating the information contained in one constraint to the neighboring constraints. Such techniques reduce the size of the solution space that needs to be explored [9]. CP has been used to solve a wide range of problems in the literature. Hande [10] proposed a CP model for the *Open Vehicle Routing problem with Heterogeneous Vehicle Fleet* (HFOVRP). In [10] the CP model is compared with a mixed-integer linear programming (MILP) model of the HFOVRP, and they showed that the CP model is effective for providing good-quality solutions for small-sized instances of the HFOVRP in short computational times compared to the MILP model. Kasapidis et al. [11] presented a MILP model and a CP model for the *Multi-Resource Flexible Job-Shop Scheduling problem with Arbitrary Precedence Graphs*. The computational experiments conducted in [11] has shown that the CP model is more effective and achieves the best

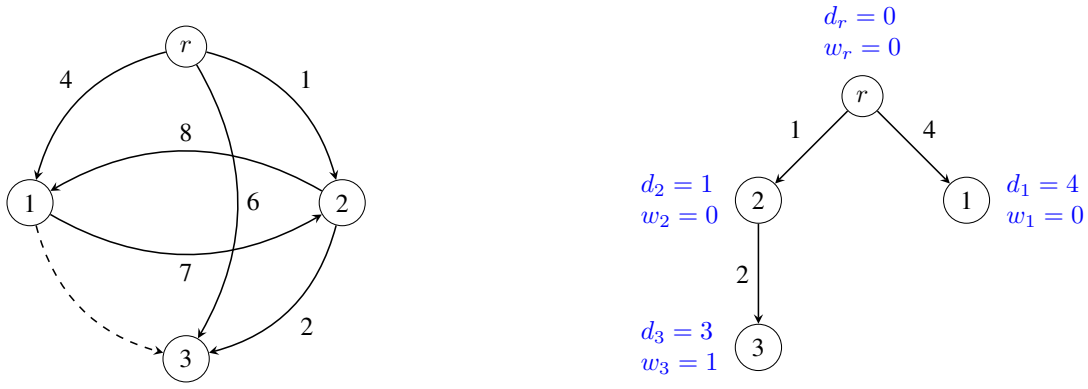

Fig. 1: Example of an instance solved as a PCMCA-WT. The graph on left shows the instance with its respective arc costs, and the precedence relationship $(1, 3) \in R$ marked as a dashed arrow. The graph on the right shows an optimal PCMCA-WT solution of cost 8.

results compared to the MILP model, although more time-consuming on some instances. Kirac et al. [12] proposed a CP approach for solving the *Team Orienteering problem with Time Windows and Mandatory Visits*, and they showed that the CP-based approach finds 99 of the best-known solutions and explores 64 new best-known solutions for the benchmark instances. Kizilay et al. [13] proposed a novel CP model for the *Mixed-Blocking Permutation Flow Shop Scheduling problem with Batch Delivery* that minimizes the total tardiness and batch cost. The results of their study has shown that due to the complexity of the problem, the developed CP model can solve only small-sized instances in reasonable computational time. Montemanni and Dell'Amico [14] proposed a CP model for the *Parallel Drone Scheduling Traveling Salesman problem*, and showed that by exploiting multi-threading computation, the method was able to optimally solve all the instances considered in the literature.

The *Precedence-Constrained Minimum-Cost Arborescence* (PCMCA) problem is an $\mathcal{NP}$-hard problem [15] that was first introduced by Dell'Amico et al. [16]. The PCMCA problem is an extension to the MCA problem, in which precedence constraints must be satisfied as follows. Given a set R of ordered pairs of vertices, then for each precedence relationship $(s, t) \in R$, a path in the solution which covers both s and t, must visit vertex s before visiting vertex t. The objective is to find an arborescence of minimum total cost satisfying the precedence constraints. The PCMCA problem has applications in the design of commodity distribution networks where certain paths are not allowed in the network due to logistical constraints [16]. Several MILP models of the problem were proposed in [15], [16], [17].

The *Precedence-Constrained Minimum-Cost Arborescence Problem with Waiting-Times* (PCMCA-WT) is an $\mathcal{NP}$-hard problem that was recently introduced by Chou et al. [15]. The PCMCA-WT is a variation on the PCMCA problem characterized by the following differences. Given arc costs indicating the time required to traverse an arc, suppose there is a flow which starts at the root vertex r, that must reach every vertex in an arborescence T. For each precedence relationship $(s, t) \in R$, the flow must enter vertex s at the same time step, or before entering vertex t, which means that the flow can stop at any vertex and wait. The waiting time at vertex t is defined as the difference between the time at which the flow enters s and the time at which the flow reaches t. The objective of the problem is to find an arborescence T of minimum total cost, plus total waiting times, where the flow never enters s after entering t for all $(s, t) \in R$. Several MILP models for solving the problem were proposed in [15].

The PCMCA-WT problem can be formally defined as follows. Given a directed graph $G = (V, A, R, r)$, where $V = \{1, \ldots, n\}$ is the set of vertices, $A \subseteq V \times V$ is the set of arcs, $R \subset V \times V$ is the set of precedence relationships, and $r \in V$ is the root of the arborescence. Let c_{ij} be a cost associated with each arc $(i, j) \in A$ which represents the time required for the flow to travel from vertex i to vertex j. Let d_j be the time step at which the flow enters vertex $j \in V$, and let w_j be the waiting time at vertex $j \in V$. The objective of the problem is to find an arborescence T rooted at vertex r, that has a minimum total cost plus total waiting time, where the flow never enters t before entering s for all $(s, t) \in R$ (i.e. $d_t \geq d_s$ for all $(s, t) \in R$).

Figure 1 presents an example of an instance solved as a PCMCA-WT. The instance graph (left graph) shows the precedence relationship $(1, 3) \in R$ marked as a dashed arrow, while the solution graph (right graph) shows an optimal solution of that instance, with the corresponding d_i and w_i value written next to each vertex. The graph on the right shows an optimal PCMCA-WT solution of cost 8 (sum of all the arcs cost plus total waiting time at each vertex), with a resulting waiting time of value 1 at vertex 3, since $d_1 = 4$, while $d_3 = 3$ and $(1, 3) \in R$.

The rest of this paper is organized as follows. Section II introduces the MILP model used in this study. Section III introduces a CP model that extends the MILP model introduced in Section II by introducing redundant constraints for a subset of the original inequalities and describing them in terms of

CP constructs, in order to further exploit the capabilities of the CP solver. Section IV summarizes computational results, while some conclusions are outlined in Section V.

II. A Mixed Integer Linear Programming Model

A polynomial-size MILP model for the PCMCA-WT was recently proposed by Dell'Amico et al. [18]. The model extends a classical formulation for the MCA problem [19], through the addition of precedence-enforcing constraints. The precedence-enforcing constraints detect a precedence violating path by propagating a value along all the paths of the solution starting from t for all $(s,t) \in R$ [16], [17].

A different version of the model that contains a smaller number of variables and constraints was also proposed in [18]. The reduction is achieved by exploiting the special property of the PCMCA-WT, that is for any precedence relationship $(s,t) \in R$, the flow must enter vertex t at the same time step or after entering vertex s. This implies that it is possible to remove a precedence relationship $(s,t) \in R$ when the input graph does not contain a zero-cost path that starts from t and ends in s. The reduced model for the PCMCA-WT proposed in [18] is summarized as follows. For further details the reader can refer to [18].

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_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. Let $P_{ij} \subset A$ be a simple directed path that starts from i and ends at j, and let $c(P_{ij}) = \sum_{(i,j)\in P} c_{ij}$ be the cost of that path. For each $s \in V$, let $V_s = \{t \in V \backslash \{r\} \,|\, \exists\, (s,t) \in R, c(P_{ts}) = 0\}$. The PCMCA-WT can be formulated as the following MILP model.

$$\text{minimize} \quad \sum_{(i,j)\in A} c_{ij} x_{ij} + \sum_{i \in V} w_i \tag{1}$$

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

$$y_i - y_j + 1 \leq n(1 - x_{ij}) \qquad \forall (i,j) \in A : j \neq r \tag{3}$$

$$u_s^t = 0 \qquad \forall (s,t) \in R : t \in V_s \tag{4}$$

$$u_t^t = 1 \qquad \forall t \in V_s \tag{5}$$

$$u_j^t - u_i^t - x_{ij} \geq -1 \quad \forall (s,t) \in R : t \in V_s, (i,j) \in A \tag{6}$$

$$d_r = 0 \tag{7}$$

$$w_r = 0 \tag{8}$$

$$d_j \geq d_i - M + (M + c_{ij}) x_{ij} \qquad \forall (i,j) \in A \tag{9}$$

$$w_j \geq d_j - d_i - M + (M - c_{ij}) x_{ij} \qquad \forall (i,j) \in A \tag{10}$$

$$d_t \geq d_s \qquad \forall (s,t) \in R \tag{11}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A \tag{12}$$

$$y_i \geq 0 \qquad \forall i \in V \tag{13}$$

$$u_j^t \geq 0 \qquad \forall t \in V_s, j \in V \tag{14}$$

$$d_i, w_i \in \mathbb{R}^+_{\leq M} \qquad \forall i \in V \tag{15}$$

The set of constraints (2) impose that every vertex excluding the root must have exactly one parent. Constraints (3) are the subtour elimination constraints, which enforce that any feasible solution is acyclic. The set of constraints (2) and (3) guarantee that any feasible solution is an arborescence rooted at vertex $r \in V$. Constraints (4) and (5) fix the values of u_s^t and u_t^t to 0 and 1 respectively, for all $(s,t) \in R$, where $t \in V_s$. Constraints (6) impose that if $x_{ij} = 1$ then $u_j^t \geq u_i^t$ (see Figure 2 for further explanation). Constraint (7) sets the time step at which the flow enters the root to 0. Constraint (8) sets the waiting time at the root r to be equal to 0. Constraints (9) impose that when arc $(i,j) \in A$ is selected to be part of the solution, then the flow enters vertex j at a time step that is greater than or equal to the time step at which the flow enters vertex i plus c_{ij}. Constraints (10) 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 (11) 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 (12)-(15) define the domain of the variables, and M is an upper bound on the value of an optimal solution.

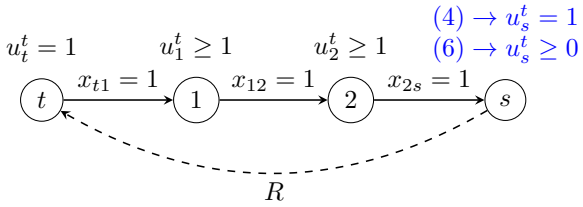

Fig. 2: An example on how a precedence relationship $(s,t) \in R$ can be enforced by propagating the value of u_t^t along every path starting from t, and if the solution contains a path from t to s, then we are propagating a value of one to vertex s and imposing that $u_s^t \geq 1$. However, we enforce $u_s^t = 0$, and therefore the solution violates the precedence relationship $(s,t) \in R$.

III. A New Constraint Programming Model

The CP solver used in this study, CP-SAT [20] is a solver that utilizes integer programming techniques (linear relaxation, presolve, cuts, and branching heuristics) to enhance its performance [21] and has recently been shown to successfully deal with different combinatorial optimization problems [22], [23]. Furthermore, the computational results in Section IV show that for the model considered in this work, the CP solver outperforms the MILP solver on a subset of the instances considered in terms of achieved average optimality gap, solution time, and the quality of the solutions obtained. Therefore, we introduce a CP model in this section that extend the MILP model introduced in Section II by adding the set of constraints

(3) and (6) formulated as logical constraints, and merging the two sets of constraints (9) and (10) into one set of logical constraints. By doing so, we further exploit the capabilities of the CP solver. Since the CP solver used utilizes integer programming techniques, it is beneficial to include both the logical and linear form of the constraints in the model, so that when the logical constraint is not enforced by the SAT solver (i.e. the logical constraint is not included in the model by the solver), their equivalent linear constraint is included in the program when computing its linear relaxation.

Using a set of implication constraints which enforce the implied constraint when the value of the variable is true, the MILP model introduced in Section II can be extended using the following set of constraints.

$$x_{ij} \implies y_j = y_i + 1 \qquad \forall (i,j) \in A : j \neq r \quad (16)$$

$$x_{ij} \implies u_j^t \geq u_i^t \qquad \forall t \in V_t, j \in V \backslash \{r\} \quad (17)$$

$$x_{ij} \implies d_j = d_i + w_j + c_{ij} \qquad \forall (i,j) \in A \quad (18)$$

Constraints (16) are the subtour elimination constraints modelling the nonlinear relationship $y_j = (y_i + 1)x_{ij}$. Constraints (17) are the precedence-enforcing constraints modeling the nonlinear relationship $u_j^t \geq u_i^t x_{ij}$. Constraints (18) combine the two constraints (9) and (10) into a single equality constraint that model the nonlinear relationship $(d_j - d_i - w_j - c_{ij})x_{ij} = 0$. 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. In this study we only consider instances with integer coefficients. However, the interested reader can refer to [14] where the authors show how changing the k value can affect the computation time.

IV. Experimental Results

The computational experiments are based on the benchmark instances of TSPLIB [24], SOPLIB [25], [26], and COMPILERS [27], originally proposed for the *Sequential Ordering Problem* (SOP) [28], [29], [30]. The benchmark instances are the same instances previously adopted in [15], [18] for the PCMCA-WT with the following characteristics. The benchmark sets contain a total of 116 instances (81 open instances) ranging in size between 9 and 700 vertices, with an average of 248 vertices. Finally, all instances have integer coefficients (i.e. the weight of the arcs of the cost graph is integer). All the experiments are performed on an Intel Xeon Platinum 8375C processor with 8 cores running at 2.9 GHz with 16 GB of RAM. For all instances an upper bound on the value of the optimal solution (i.e. M), is set to the value of the solution cost of solving the instance as a SOP, using a nearest neighbor algorithm [27]. This is a valid upper bound for the cost of the optimal solution of the PCMCA-WT, being a feasible solution for the SOP a simple directed path that includes all the vertices of the graph, such that t never precede s for all $(s,t) \in R$. This implies that $d_t \geq d_s$ for all $(s,t) \in R$, with a waiting time equal to zero at each vertex by definition.

The computational results are generated using two solvers: a *MILP Solver* and a *CP Solver*. The MILP Solver is CPLEX v12.8 [31], and is run with 8 thread standard B&C algorithm, with the two parameters *NodeSelect* and *MIP emphasis* are set to *BestBound* and *MIPEmphasisOptimality* respectively. The CP Solver is Google OR-Tools [20] v9.5 CP-SAT solver, and is run with its default parameters 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. For the rest of this section we will be referring to the MILP model introduced in Section II as *BM* (Basic Model), while the CP model introduced in Section III will be referred to as *RM* (Reinforced Model).

Tables I, II and III show the complete results of each model and solving method, 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. The best-known solutions are obtained from the results appeared in [18], generated using the same computational setup and configuration used in this study. For each model solved with the corresponding solver, we report the following columns. Columns *LB* and *UB* report the lower/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 *Branches* reports the number of branches created in the search-decision tree, and is only reported when the models are solved with the CP Solver. Finally, column *Time [s]* 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 found.

A. Multi-threading Computation

The performance of CP solvers can often be greatly improved by the use of multi-threading computation, usually more than MILP solvers due to the different approaches used to solve the mathematical model. In this section, we assess the effect of multi-threading on the performance of the CP Solver and MILP Solver at solving the model introduced in Section II. The four instances *ft53.1, prob.42, ESC78*, and *jpeg.4753.54* were selected as both solvers are able to optimally solve those instances within the time limit using 8 threads.

Figure 3 reports the time required to optimally solve the different instances considered using a number of threads

TABLE I: Computational results for TSPLIB instances.

Instance				MILP Solver				CP Solver									
				BM				BM					RM				
Name	Size	$\rho(R)$	Best-Known	LB	UB	Gap	Time [s]	LB	UB	Gap	Branches	Time [s]	LB	UB	Gap	Branches	Time [s]
br17.10	18	0.314	44	40	44	0.091	-	44	44	0.000	56849	43.544	44	44	0.000	40829	34.568
br17.12	18	0.359	44	41	44	0.068	-	44	44	0.000	1225655	40.911	44	44	0.000	86038	36.960
ESC07	9	0.611	1906	1906	1906	0.000	0.070	1906	1906	0.000	153	0.050	1906	1906	0.000	102	0.020
ESC11	13	0.359	2174	2174	2174	0.000	0.114	2174	2174	0.000	339	0.059	2174	2174	0.000	339	0.076
ESC12	14	0.396	1138	1138	1138	0.000	0.030	1138	1138	0.000	326	0.044	1138	1138	0.000	302	0.046
ESC25	27	0.177	1158	1158	1158	0.000	1.945	1158	1158	0.000	16928	0.634	1158	1158	0.000	1451	0.578
ESC47	49	0.108	747	747	747	0.000	22.153	747	747	0.000	90693	3.008	747	747	0.000	5473	2.856
ESC63	65	0.173	56	56	56	0.000	57.347	56	56	0.000	7842	1.089	56	56	0.000	7708	1.289
ESC78	80	0.139	1196	1196	1196	0.000	257.609	1196	1196	0.000	297687	22.228	1196	1196	0.000	16383	7.453
ft53.1	54	0.082	4089	4089	4089	0.000	2023.553	4089	4089	0.000	47285	109.466	4089	4089	0.000	835469	110.951
ft53.2	54	0.094	[4161, 4284]	4161	4334	0.040	-	**4284**	**4284**	0.040	2513278	1284.742	**4284**	**4284**	0.000	237937	1522.620
ft53.3	54	0.225	[4799, 5279]	4799	5279	0.091	-	4474	5322	0.159	2596272	-	4428	5481	0.192	2275894	-
ft53.4	54	0.604	[5923, 6420]	5923	6420	0.077	-	5490	6489	0.154	2229608	-	5397	6420	0.159	2112150	-
ft70.1	71	0.036	[33101, 33298]	32827	33308	0.014	-	33105	33298	0.006	1130106	-	**33111**	33298	0.006	6020308	-
ft70.2	71	0.075	[33089, 33670]	33089	33916	0.024	-	**33261**	33824	0.017	5506591	-	33259	33670	0.012	4140282	-
ft70.3	71	0.142	[34423, 36932]	34423	38351	0.102	-	33813	**36152**	0.065	2653144	-	33773	36372	0.071	3190167	-
ft70.4	71	0.589	[36850, 36939]	36850	38771	0.050	-	35838	39706	0.097	4827165	-	35813	39897	0.102	3417417	-
rbg048a	50	0.444	263	259	264	0.019	-	263	263	0.000	14834	17.308	263	263	0.000	14896	17.679
rbg050c	52	0.459	225	225	225	0.000	36.673	225	225	0.000	8549	1.363	225	225	0.000	5436	2.196
rbg109	111	0.909	[366, 401]	366	407	0.101	-	358	402	0.109	1795755	-	357	**400**	0.108	172527	-
rbg150a	152	0.927	[463, 509]	461	509	0.094	-	454	556	0.183	153864	-	420	556	0.245	124524	-
rbg174a	176	0.929	[463, 553]	463	553	0.163	-	454	**537**	0.155	1356878	-	455	551	0.174	1226907	-
rbg253a	255	0.948	[532, 718]	532	718	0.259	-	515	672	0.234	888937	-	511	**665**	0.232	110453	-
rbg323a	325	0.928	[1009, 1891]	974	2466	0.605	-	996	1636	0.391	152276	-	889	**1544**	0.424	104927	-
rbg341a	343	0.937	[780, 1457]	761	2907	0.738	-	643	1283	0.499	640718	-	668	**1213**	0.449	1928843	-
rbg358a	360	0.886	[788, 1150]	755	2453	0.692	-	758	**1052**	0.279	219152	-	701	1130	0.380	130991	-
rbg378a	380	0.894	[678, 1126]	648	2191	0.704	-	582	**1070**	0.456	543767	-	639	1087	0.412	352856	-
kro124p.1	101	0.046	[32630, 33962]	32630	36099	0.096	-	32576	34235	0.048	422907	-	32544	34433	0.055	643136	-
kro124p.2	101	0.053	[33006, 35860]	33006	39931	0.173	-	32781	36284	0.097	10775155	-	32800	36687	0.106	12286675	-
kro124p.3	101	0.092	[34005, 42416]	34005	46764	0.273	-	33716	40958	0.177	7997226	-	33621	**40814**	0.176	527262	-
kro124p.4	101	0.496	[39333, 49590]	39333	53456	0.264	-	38268	48940	0.218	7075173	-	38333	**48035**	0.202	1783421	-
p43.1	44	0.101	[2860, 3955]	2656	3955	0.328	-	2860	3980	0.281	5139272	-	2852	3955	0.279	2323809	-
p43.2	44	0.126	[2870, 4160]	2705	4210	0.357	-	2837	4105	0.309	7941383	-	**2877**	**4020**	0.284	3390171	-
p43.3	44	0.191	[2966, 4255]	1383	4440	0.689	-	2880	4350	0.338	3165556	-	2929	4425	0.338	1828845	-
p43.4	44	0.164	[3125, 4495]	3125	4605	0.321	-	3048	4495	0.322	1603271	-	3047	4540	0.329	1316102	-
prob.100	100	0.048	[677, 738]	677	741	0.086	-	674	738	0.087	221458	-	674	**734**	0.082	178127	-
prob.42	42	0.116	171	171	171	0.000	230.506	171	171	0.000	230646	37.298	171	171	0.000	28930	78.549
ry48p.1	49	0.091	[13200, 13670]	13200	13670	0.034	-	13197	**13665**	0.034	692909	-	13157	13670	0.038	754011	-
ry48p.2	49	0.103	[13336, 14305]	13336	14305	0.068	-	**13370**	**14224**	0.060	23506295	-	13360	**14224**	0.061	16644814	-
ry48p.3	49	0.193	[13994, 15477]	13994	15840	0.117	-	13757	15477	0.111	17599209	-	13949	**15439**	0.097	554056	-
ry48p.4	49	0.588	[17180, 19495]	17180	19583	0.123	-	15867	19544	0.188	393528	-	15848	19656	0.194	380840	-
Average						0.167	263.000			0.125	2822894	111.553			0.127	1687825	129.703

TABLE II: Computational results for SOPLIB instances.

Instance				MILP Solver				CP Solver									
				BM				BM					RM				
Name	Size	$\rho(R)$	Best-Known	LB	UB	Gap	Time [s]	LB	UB	Gap	Branches	Time [s]	LB	UB	Gap	Branches	Time [s]
R.200.100.1	200	0.020	29	29	29	0.000	6.017	29	29	0.000	202206	29.711	29	29	0.000	83586	26.963
R.200.100.15	200	0.847	[589, 979]	525	1033	0.492	-	563	**972**	0.421	2282206	-	550	1035	0.469	2155929	-
R.200.100.30	200	0.957	[838, 1761]	774	1761	0.560	-	843	1714	0.508	140133	-	809	**1708**	0.526	1378303	-
R.200.100.60	200	0.991	[8861, 16197]	8861	16930	0.477	-	8057	15595	0.483	577845	-	7628	**15176**	0.497	108865	-
R.200.1000.1	200	0.020	887	887	887	0.000	15.635	887	887	0.000	83329	29.587	887	887	0.000	79196	33.442
R.200.1000.15	200	0.876	[7231, 12601]	6895	12601	0.453	-	6205	12788	0.515	5742971	-	7033	14254	0.507	161527	-
R.200.1000.30	200	0.958	[10512, 22781]	10512	22781	0.539	-	9959	24172	0.588	3764158	-	9802	**22693**	0.568	123982	-
R.200.1000.60	200	0.989	[12042, 19934]	12042	21993	0.452	-	10122	21435	0.528	114847	-	9910	21377	0.536	8963959	-
R.300.100.1	300	0.013	13	13	13	0.000	35.012	13	13	0.000	68802	45.037	13	13	0.000	161824	54.610
R.300.100.15	300	0.905	[811, 2056]	669	2259	0.704	-	749	**1865**	0.598	113431	-	**813**	2080	0.609	50986	-
R.300.100.30	300	0.970	[1157, 2590]	1102	3163	0.652	-	1132	2785	0.594	81595	-	1130	2931	0.614	74613	-
R.300.100.60	300	0.994	[991, 1865]	949	1954	0.514	-	932	2293	0.594	199836	-	953	2344	0.593	342126	-
R.300.1000.1	300	0.013	715	715	715	0.000	64.683	715	715	0.000	212106	130.421	715	715	0.000	758190	87.442
R.300.1000.15	300	0.905	[8768, 24047]	7832	24047	0.674	-	7544	**20895**	0.639	149209	-	7524	22818	0.670	3176756	-
R.300.1000.30	300	0.965	[12269, 31618]	12071	40863	0.705	-	10706	43534	0.754	5185819	-	10415	41310	0.748	428501	-
R.300.1000.60	300	0.994	[10408, 21623]	10275	25323	0.594	-	8561	25887	0.669	106918	-	8655	**21294**	0.594	684540	-
R.400.100.1	400	0.010	6	6	6	0.000	995.137	6	6	0.000	67605	129.029	6	6	0.000	4181266	76.868
R.400.100.15	400	0.927	[963, 3591]	856	22767	0.962	-	955	2488	0.616	2480098	-	849	**2830**	0.700	125348	-
R.400.100.30	400	0.978	[1084, 3061]	1010	26438	0.962	-	997	**2678**	0.628	89327	-	1001	2839	0.647	1890510	-
R.400.100.60	400	0.996	[966, 2069]	861	2652	0.675	-	666	2135	0.688	139991	-	736	2084	0.647	123574	-
R.400.1000.1	400	0.010	780	780	780	0.000	124.990	780	780	0.000	209665	76.386	780	780	0.000	56921	105.197
R.400.1000.15	400	0.930	[9976, 35160]	9083	85878	0.894	-	7804	**29188**	0.733	2821927	-	7642	34339	0.777	897078	-
R.400.1000.30	400	0.977	[12337, 57272]	11783	127290	0.907	-	9727	**34781**	0.720	120018	-	9927	48659	0.796	5057396	-
R.400.1000.60	400	0.995	[9954, 22376]	9877	36662	0.731	-	7237	24990	0.710	182264	-	7620	22521	0.662	128489	-
R.500.100.1	500	0.008	3	3	3	0.000	1881.297	3	3	0.000	1665985	1184.061	3	3	0.000	2352	66.186
R.500.100.15	500	0.945	[1250, 5508]	1018	11452	0.911	-	1044	**4700**	0.778	1977102	-	907	5370	0.831	137132	-
R.500.100.30	500	0.980	[1099, 4841]	976	14273	0.932	-	808	3687	0.781	120015	-	717	**3326**	0.784	130094	-
R.500.100.60	500	0.996	[931, 2723]	840	6357	0.868	-	560	5309	0.895	2325233		560	**2544**	0.780	4962333	-
R.500.1000.1	500	0.008	297	297	297	0.000	85.459	297	297	0.000	2551213	209.776	297	297	0.000	6327	63.832
R.500.1000.15	500	0.940	[10628, 45356]	9461	107776	0.912	-	8240	**35647**	0.769	149478	-	8445	41479	0.796	1363078	-
R.500.1000.30	500	0.981	[12694, 57330]	12694	156359	0.919	-	9458	**47859**	0.802	110401	-	9995	53765	0.814	1836393	-
R.500.1000.60	500	0.996	[8192, 20465]	8192	45696	0.821	-	6159	21832	0.718	559087	-	6159	22735	0.729	262935	-
R.600.100.1	600	0.007	1	1	55	0.982	-	1	1	0.000	95274	253.537	1	1	0.000	63402	118.1845
R.600.100.15	600	0.950	[938, 2443]	845	4044	0.791	-	549	3942	0.861	1233613		662	**2399**	0.724	2763030	-
R.600.100.30	600	0.985	[1099, 6467]	1099	18932	0.942	-	740	6868	0.892	1649053		789	**6346**	0.876	2955997	-
R.600.100.60	600	0.997	[778, 2494]	778	25214	0.969	-	538	3395	0.842	766227	-	538	**2833**	0.810	7723228	-
R.600.1000.1	600	0.007	322	322	322	0.000	140.645	322	322	0.000	5796029	373.208	322	322	0.000	634	81.753
R.600.1000.15	600	0.945	[10915, 65039]	10915	121877	0.910	-	9401	62114	0.849	1003612		9875	**53937**	0.817	832443	-
R.600.1000.30	600	0.984	[12431, 48775]	12431	190145	0.935	-	9356	73581	0.873	1242031	-	10008	**43929**	0.772	699944	-
R.600.1000.60	600	0.997	[8162, 42652]	8162	75269	0.892	-	6908	44310	0.844	1368338		6908	**40077**	0.828	1640120	-
R.700.100.1	700	0.006	2	-	-	-	-	2	2	0.000	4624	105.444	2	2	0.000	1345	75.207
R.700.100.15	700	0.957	[972, 2759]	972	5718	0.830	-	655	5914	0.889	909974	-	753	2995	0.749	631765	-
R.700.100.30	700	0.987	[983, 2531]	983	4218	0.767	-	588	2531	0.768	335440	-	756	2531	0.701	143114	-
R.700.100.60	700	0.997	[555, 1598]	555	1854	0.701	-	383	1598	0.760	316660	-	383	1598	0.760	236953	-
R.700.1000.1	700	0.006	611	611	616	0.008	-	611	611	0.000	31507	150.460	611	611	0.000	30830	113.285
R.700.1000.15	700	0.956	[5136, 6315]	5136	7145	0.281	-	2787	61078	0.954	365432		4316	6315	0.317	128446	-
R.700.1000.30	700	0.986	[4827, 6115]	4827	6981	0.309	-	2658	6200	0.571	294568	-	3906	6115	0.361	223725	-
R.700.1000.60	700	0.997	[2997, 5357]	2997	5842	0.487	-	1913	**5331**	0.641	284690	-	2022	5379	0.624	176457	-
Average						0.577	372.097			0.531	1047748	226.388			0.505	1211365	75.247

TABLE III: Computational results for COMPILERS instances.

Instance				MILP Solver				CP Solver									
				BM				BM					RM				
Name	Size	$\rho(R)$	Best-Known	LB	UB	Gap	Time [s]	LB	UB	Gap	Branches	Time [s]	LB	UB	Gap	Branches	Time [s]
gsm.153.124	126	0.970	[280, 311]	269	311	0.135	-	276	311	0.113	3919288	-	**283**	311	0.090	345832	-
gsm.444.350	353	0.990	[2456, 4310]	2405	4856	0.505	-	**2482**	4625	0.463	127304	-	2453	**4214**	0.418	207341	-
gsm.462.77	79	0.840	[419, 465]	402	477	0.157	-	419	471	0.110	3145801	-	419	466	0.101	4649930	-
jpeg.1483.25	27	0.484	87	87	87	0.000	18.556	87	87	0.000	1278	0.150	87	87	0.000	966	0.173
jpeg.3184.107	109	0.887	[518, 656]	510	715	0.287	-	**520**	686	0.242	2786312	-	517	**623**	0.170	2220962	-
jpeg.3195.85	87	0.740	[23, 25]	17	25	0.320	-	23	25	0.080	488728	-	22	25	0.120	506891	-
jpeg.3198.93	95	0.752	[181, 188]	180	188	0.043	-	181	188	0.037	2762954	-	181	188	0.037	1116829	-
jpeg.3203.135	137	0.897	[629, 750]	618	751	0.177	-	622	809	0.231	273519	-	629	**709**	0.113	258220	-
jpeg.3740.15	17	0.257	33	33	33	0.000	0.839	33	33	0.000	532	0.058	33	33	0.000	490	0.069
jpeg.4154.36	38	0.633	90	90	90	0.000	60.924	90	90	0.000	5347	1.010	90	90	0.000	7836	1.855
jpeg.4753.54	56	0.769	164	164	164	0.000	1790.269	164	164	0.000	234891	33.849	164	164	0.000	55605	214.247
susan.248.197	199	0.939	[805, 1320]	802	1370	0.415	-	**810**	**1200**	0.325	1205085	-	807	1557	0.482	1065206	-
susan.260.158	160	0.916	[598, 897]	573	938	0.389	-	586	969	0.395	1146515	-	588	934	0.370	1170422	-
susan.343.182	184	0.936	[636, 776]	622	776	0.198	-	641	817	0.215	1478092	-	639	796	0.197	1208246	-
typeset.10192.123	125	0.744	[293, 379]	282	379	0.256	-	**300**	379	0.208	2212422	-	292	393	0.257	425307	-
typeset.10835.26	28	0.349	[110, 111]	100	112	0.107	-	109	111	0.018	925171	-	108	111	0.027	841850	-
typeset.12395.43	45	0.518	146	141	146	0.034	-	146	146	0.000	369091	475.072	146	146	0.000	82439	250.703
typeset.15087.23	25	0.557	97	97	97	0.000	29.118	97	97	0.000	1469	0.228	97	97	0.000	814	0.336
typeset.15577.36	38	0.555	125	125	125	0.000	43.164	125	125	0.000	55895	2.485	125	125	0.000	18202	11.596
typeset.16000.68	70	0.658	[79, 80]	66	80	0.175	-	74	80	0.075	2083457	-	74	80	0.075	1029470	-
typeset.1723.25	27	0.245	60	60	60	0.000	86.068	60	60	0.000	1550	0.260	60	60	0.000	2537	0.362
typeset.19972.246	248	0.993	[1525, 2509]	1452	2509	0.421	-	1514	3001	0.496	591545	-	1515	2692	0.437	1047705	-
typeset.4391.240	242	0.981	[1154, 1905]	1137	2476	0.541	-	**1172**	1379	0.150	1261056	-	1165	**1324**	0.120	147004	-
typeset.4597.45	47	0.493	154	151	154	0.019	-	154	154	0.000	465417	497.584	154	154	0.000	127268	380.478
typeset.4724.433	435	0.995	[2679, 6131]	2673	6131	0.564	-	**2701**	**5738**	0.529	98263	-	2676	6275	0.574	521523	-
typeset.5797.33	35	0.748	113	113	113	0.000	28.504	113	113	0.000	1902	0.600	113	113	0.000	1286	0.752
typeset.5881.246	248	0.986	[1406, 2084]	1396	2426	0.425	-	1426	**2067**	0.310	207231	-	1417	2082	0.319	164597	-
Average						0.191	257.180			0.148	957412	101.129			0.145	637955	86.057

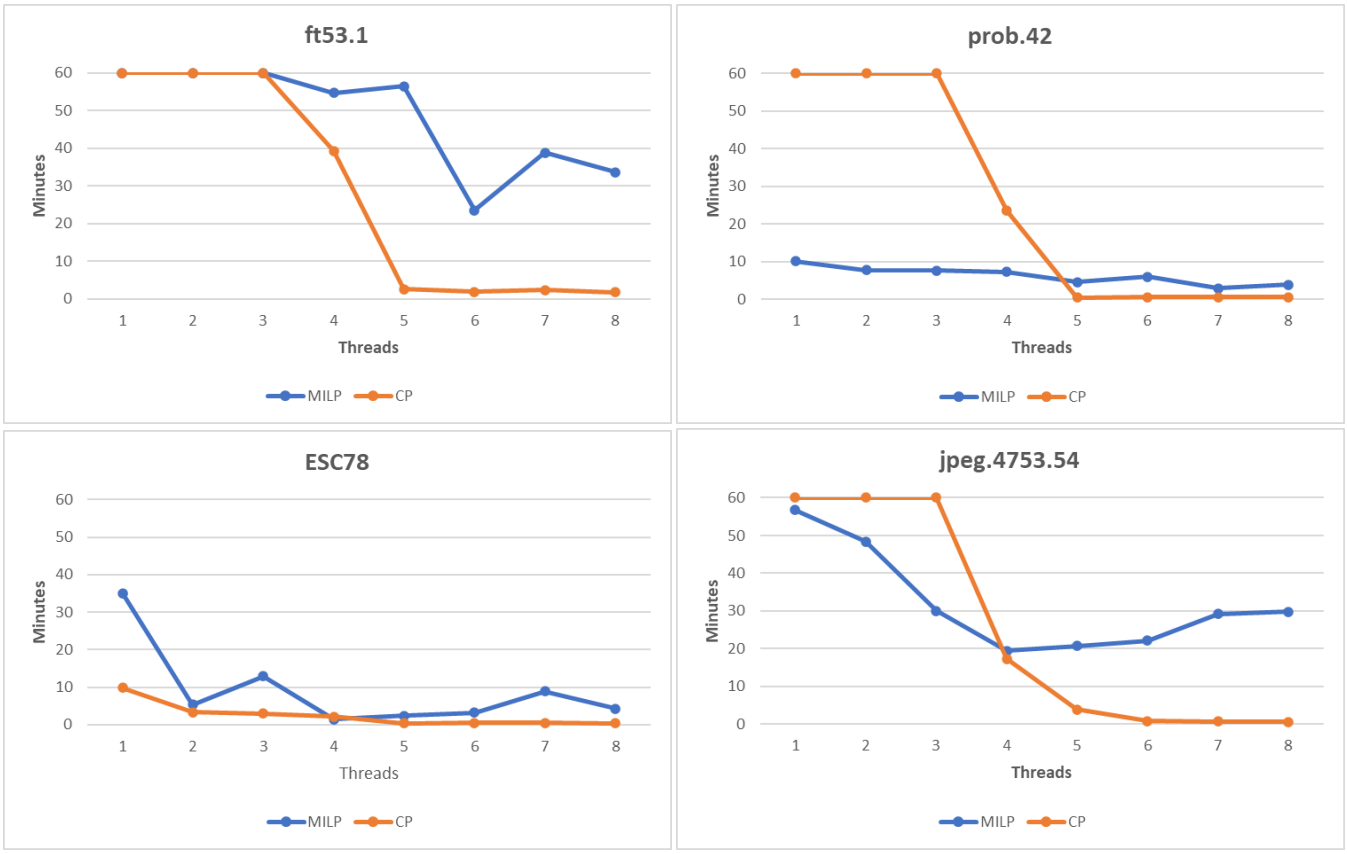

Fig. 3: Time required by the MILP Solver and CP Solver to optimally solve different instances with different number of threads. A time of 60 minutes reported means that the respective solver was not able to optimally solve the instance within the time limit.

between 1 and 8. In the figure, a time of 60 minuets reported means that the respective solver was not able to optimally solve the instance within the time limit of one hour.

The results reported in Figure 3 show that the CP Solver substantially benefits from the use of multi-threading computation. Furthermore, the results show that the CP Solver is not able to optimally solve three out of four instances within the time limit when less than four threads are allocated for the solver. However, when allocating four or more threads, the CP Solver is able to optimally solve those instances. Furthermore, a drastic change in performance can be observed between four and five threads, reaching a speedup up to 93.5%. On the other hand, the MILP Solver does not seem to benefit as much from multi-threading for the instances considered, possibly due to the overhead of task distribution, and the waiting time incurred by the variety of methods run in parallel. Furthermore, we can notice less consistent gain when increasing the number of threads used by the MILP Solver compared to the CP Solver. It should be noted that the differences between the two solvers might be less extreme when more challenging instances are considered, but this is difficult to investigate as most instances are hard to solve optimally, even with longer computational time limit (hours) is allowed, and with eight threads allocated for the solvers.

In conclusion, the CP Solver appears to greatly benefit from multi-threading computation; therefore, all the experiments reported in this section were run on eight cores.

B. *Analysis of the Results*

In this section, we first compare and discuss the results achieved by the MILP and CP Solvers by solving the model *BM*. We then compare and discuss the results achieved by the CP Solver by solving the models *BM* and *RM*.

TABLE IV: Summary of the results achieved by solving the model *BM* with the MILP Solver and CP Solver.

	MILP Solver	CP Solver
Average optimality gap	0.340	0.301
Average solution time	297.6	89.7
New best-known lower bounds	0	9
New best-known upper bounds	0	19
New optimal solution	0	1

Table IV summarizes the results of solving the model *BM* by the MILP Solver and CP Solver, where we report the following. The *Average optimality gap* is computed with respect

to all the instances where both solvers find a feasible/optimal solution before reaching the time limit when solving the model *BM*. The *Average solution time* is computed on all the instances that are solved optimally by the both solvers. The *New best-known lower bounds* and *New best-known upper bounds* rows report the number of instances where solving the model by each solver resulted in an improved lower or upper bound. Finally, *New optimal solution* row reports the number of instances where an optimal solution is found for an instance that was previously open, by each solver.

Considering the model *BM*, the MILP Solver achieves an average optimality gap of 0.340 across all the instances, but fails to solve a single instance (marked bold in the table) as it runs out of memory while solving the linear relaxation of the model. On the other hand, the CP Solver achieves an average optimality gap of 0.301 (a 11.5% improvement) when excluding the instance that is not solved by the MILP Solver, and an average optimality gap of 0.298 (a 12.4% improvement) across all the instances. By further inspecting the results, we notice that the CP Solver achieves a smaller average optimality gap within the time limit for instances with density less than 0.85 and size smaller than 400.

For a total of 27 instances that are optimally solved by both solvers, the MILP Solver has an average solution time of 297.6 seconds, while the CP Solver has an average solution time of 89.7 seconds (a 69.9% improvement). We should note that the CP Solver generally finds the optimal solution in less time compared to the MILP Solver on small to medium sized instances.

Finally, out of a total of 81 open instances, the CP Solver is able to find an improved lower bound for 9 instances (11.1%), an improved upper bound for 19 instances (23.5%), and finds the optimal solution of one instance that was previously open. On the other hand, the MILP Solver is not able to improve the best-known solution of any instance. Based on the experiments performed on the model *BM* presented in Section II, we can conclude that the CP Solver has an overall better performance at solving the given MILP model.

TABLE V: Summary of the results achieved by solving each model with the CP Solver.

	BM	RM
Average optimality gap	0.298	0.287
Average solution time	146.9	99.4
New best-known lower bounds	9	4
New best-known upper bounds	19	27
New optimal solution	1	1

The rest of this section discusses the results achieved by the CP Solver by solving the two models *BM* and *RM*. The results are summarized in Table V where we report the following. The *average optimality gap* reports the Average optimality gap of all the instances where the solver finds a feasible/optimal solution before reaching the time limit by solving both models. The *Average solution time* reports the average solution time in seconds of all the instances that are solved optimally by the solver when solving both models. The *New best-known lower bounds* and *New best-known upper bounds* rows report the number of instances where solving each model resulted in an improved lower/upper bound. Finally, *New optimal solution* report the number of instances where an optimal solution is found for an instance that was previously open.

In terms of achieved average optimality gap, the CP Solver achieves an average optimality gap of 0.287 (a 3.7% improvement) when solving the model *RM*, compared to solving the model *BM*. Furthermore, for a total of 36 instances that are solved optimally when solving both models, the CP Solver generates 57.9% less branches in the search-decision tree when solving the model *RM* compared to solving the model *BM*. By further inspecting the results, we notice that the CP Solver achieves a smaller average optimality gap within the time limit when solving the model *RM* for instances with density less than 0.89 and size less than 500, which means that the CP Solver performs better on a larger subset of the instances compared to solving the model *BM*.

For a total of 36 instances that are solved optimaly by the CP Solver when solving both models, the CP Solver has an average solution time of 146.9 seconds when solving the model *BM*, and an average solution time of 99.4 seconds (a 32.4% improvement) when solving the model *RM*.

Finally, out of a total of 81 open instances, when solving the model *BM* the CP Solver finds an improved lower bound for 9 instances (11.1%), an improved upper bound for 19 instances (23.5%), and finds the optimal solution for one instances that was previously open. On the other hand, when solving the model *RM* the CP Solver finds an improved lower bound for 4 instances (4.9%), an improved upper bound for 27 instances (33.3%), and finds the optimal solution for the same instance that was previously open. Based on the computational experiments and the improvements in the results achieved by the CP Solver when solving the model *RM*, we can conclude that duplicating the constraints can indeed improve the performance of the solver for the given model.

V. Conclusions

The computational experiments has shown that the CP Solver outperforms the MILP Solver at solving instances with sizes up to 500 with precedence relationships density that is less than 0.89. Furthermore, the CP Solver achieves a smaller average optimality gap and solution time compared to the MILP Solver. By adding constraint programming constructs to the MILP model, we were able to further exploit the capabilities of the CP Solver, and improve its performance at solving the instances. In terms of solution quality, and out of a total of 81 open instances, the CP Solver was able to find the optimal solution to an instance that was previously open, provide new best-known lower bounds for 13 instances, and establish new best-known solution for 46 instances. Based on the computational experiments performed, we have shown that the CP Solver performs better on average for the given models. Furthermore, duplicating constraints by defining them in their linear form and logical form further pushes the performance

of the CP Solver. Future work will consider investigating new valid constraints/inequalities for the PCMCA-WT that can be used within a constraint programming paradigm to further utilize its potential.

REFERENCES

[1] J. Edmonds, "Optimum branchings," *Journal of Research of the National Bureau of Standards*, vol. B 71, no. 4, pp. 233–240, 1967.

[2] Y. J. Chu and T. Liu, "On the shortest arborescence of a directed graph," *Scientia Sinica*, vol. 14, pp. 1396–1400, 1965.

[3] H. N. Gabow, Z. Galil, T. Spencer, and R. E. Tarjan, "Efficient algorithms for finding minimum spanning trees in undirected and directed graphs," *Combinatorica*, vol. 6, no. 2, pp. 109–122, 1986.

[4] M. Fischetti and D. Vigo, "A branch-and-cut algorithm for the resource-constrained minimum-weight arborescence problem," *Networks: An International Journal*, vol. 29, no. 1, pp. 55–67, 1997.

[5] L. Gouveia and M. J. Lopes, "The capacitated minimum spanning tree problem: On improved multistar constraints," *European Journal of Operational Research*, vol. 160, no. 1, pp. 47–62, 2005.

[6] G. Fertin, J. Fradin, and G. Jean, "Algorithmic aspects of the maximum colorful arborescence problem," in *Theory and Applications of Models of Computation. TAMC 2017*. Springer, 2017, pp. 216–230.

[7] N. Guttmann-Beck and R. Hassin, "On two restricted ancestors tree problems," *Information processing letters*, vol. 110, no. 14-15, pp. 570–575, 2010.

[8] S. C. Brailsford, C. N. Potts, and B. M. Smith, "Constraint satisfaction problems: Algorithms and applications," *European journal of operational research*, vol. 119, no. 3, pp. 557–581, 1999.

[9] F. Rossi, P. Van Beek, and T. Walsh, "Constraint programming," *Foundations of Artificial Intelligence*, vol. 3, pp. 181–211, 2008.

[10] H. Öztop, "A constraint programming model for the open vehicle routing problem with heterogeneous vehicle fleet," in *Towards Industry 5.0: Selected Papers from ISPR2022, October 6–8, 2022, Antalya*. Springer, 2023, pp. 345–356.

[11] G. Kasapidis, S. Dauzère-Pérèz, D. Paraskevopoulos, P. Repoussis, and C. Tarantilis, "On the multi-resource flexible job-shop scheduling problem with arbitrary precedence graphs," *Production and Operations Management*, 2023.

[12] E. Kirac, R. Gedik, and F. Oztanriseven, "Solving the team orienteering problem with time windows and mandatory visits using a constraint programming approach," *International Journal of Operational Research*, vol. 46, no. 1, pp. 20–42, 2023.

[13] D. Kizilay, Z. A. Çil, H. Öztop, and İ. Bağcı, "A novel mathematical model for mixed-blocking permutation flow shop scheduling problem with batch delivery," in *Towards Industry 5.0: Selected Papers from ISPR2022, October 6–8, 2022, Antalya*. Springer, 2023, pp. 453–461.

[14] R. Montemanni and M. Dell'Amico, "Solving the parallel drone scheduling traveling salesman problem via constraint programming," *Algorithms*, vol. 16, no. 1, p. 40, 2023.

[15] X. Chou, M. Dell'Amico, J. Jamal, and R. Montemanni, "Precedence-constrained arborescences," *European Journal of Operational Research*, vol. 307, no. 2, pp. 575–589, 2022.

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

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

[18] M. Dell'Amico, J. Jamal, and R. Montemanni, "Compact models for the precedence-constrained minimum-cost arborescence problem with waiting-times," *Annals of Operations Research. Submitted*, 2023.

[19] N. Kamiyama, "Arborescence problems in directed graphs: Theorems and algorithms," *Interdisciplinary information sciences*, vol. 20, no. 1, pp. 51–70, 2014.

[20] Google, "Google OR-Tools," 2015, [last accessed 7-March-2023]. [Online]. Available: https://developers.google.com/optimization

[21] L. Perron, "CP-SAT over CBC for MIP, is it worthwhile?" 2020, [last accessed 7-March-2023]. [Online]. Available: https://or.stackexchange.com/questions/4119/cp-sat-over-cbc-for-mip-is-it-worthwhile

[22] R. Montemanni, G. H. Carraretto, U. J. Mele, and L. M. Gambardella, "A constraint programming model for the b-coloring problem," *International Conference on Industrial Engineering and Applications, to appear*, 2023.

[23] R. Montemanni and M. Dell'Amico, "Solving the parallel drone scheduling traveling salesman problem via constraint programming," *Algorithms*, vol. 16, no. 1, p. 40, 2023.

[24] G. Reinelt, "TSPLIB–A travelling salesman problem library," *ORSA journal on computing*, vol. 3, no. 4, pp. 376–384, 1991.

[25] R. Montemanni, D. H. Smith, and L. M. Gambardella, "A heuristic manipulation technique for the sequential ordering problem," *Computers & Operations Research*, vol. 35, no. 12, pp. 3931–3944, 2008.

[26] R. Montemanni, D. H. Smith, A. E. Rizzoli, and L. M. Gambardella, "Sequential ordering problems for crane scheduling in port terminals," *International Journal of Simulation and Process Modelling*, vol. 5, no. 4, pp. 348–361, 2009.

[27] G. Shobaki and J. Jamal, "An exact algorithm for the sequential ordeing problem and its application to switching energy minimization in compilers," *Computational Optimizations and Applications*, vol. 61, no. 2, pp. 343–372, 2015.

[28] N. Ascheuer, N. Jünger, and G. Reinelt, "A branch & cut algorithm for the asymmetric traveling salesman problem with precedence constraints," *Computational Optimization and Applications*, vol. 17, no. 1, pp. 61–84, 2000.

[29] V. Papapanagiotou, J. Jamal, R. Montemanni, G. Shobaki, and L. M. Gambardella, "A comparison of two exact algorithms for the sequential ordering problem," in *IEEE Conference on Systems Process and Control (ICSPC)*, 2015.

[30] J. Jamal, G. Shobaki, V. Papapanagiotou, L. M. Gambardella, and R. Montemanni, "Solving the sequential ordering problem using branch and bound," in *IEEE Symposium Series on Computational Intelligence (SSCI)*, 2017.

[31] IBM, "IBM CPLEX Optimizer," 1988, [last accessed 7-March-2023]. [Online]. Available: https://www.ibm.com/de-de/analytics/cplex-optimizer