DECOMPOSITION-BASED EFFICIENT HEURISTIC FOR SCHEDULING

DRAGUTIN OSTOJIĆ[1], ANDRIJA UROŠEVIĆ[2], TATJANA DAVIDOVIĆ[3], TATJANA JAKŠIĆ KRÜGER[3] AND DUŠAN RAMLJAK[4]

[1] Faculty of Science, University of Kragujevac, Kragujevac, dragutin.ostojic@pmf.kg.ac.rs
[2] Faculty of Mathematics, University of Belgrade, Belgrade, andrija.urosevic@matf.bg.ac.rs
[3] Mathematical Institute, Serbian Academy of Sciences and Arts, Belgrade, {tanjad, tatjana}@mi.sanu.ac.rs
[4] School of Professional Graduate Studies at Great Valley, The Pennsylvania State University, Malvern, dusan@psu.edu

Abstract: *We develop Decomposition-based Iterative Stochastic Transformation (DIST), an efficient heuristic algorithm for a well-studied hard combinatorial optimization problem, scheduling independent jobs on identical machines, known as $P||C_{max}$. DIST explores the relationship between $P||C_{max}$, Bin Packing Problem, and Multiple Subset Sum Problem to provide high-quality solutions. DIST relies on a partitioning strategy and iteratively performs nondeterministic transformations of current solution thus it represents a stohastic search algorithm. It has a built-in mechanism to guarantee the optimality of the provided solution and, given enough time, DIST can always solve a given instance to optimality. However, DIST exhibits very good performance within short execution time, as it is demonstrated by the experimental evaluation on the popular benchmark sets of instances.*

Keywords: *combinatorial optimization, scheduling independent jobs on identical machines, problem duality, stochastic search, lower and upper bounds*

1. INTRODUCTION

Recently, authors often use some ad hoc approaches when developing optimization algorithms. For example, they formulate a Mixed-Integer Programming (MIP) model, employ an off-the-shelf solver or use some standard paradigms (evolutionary algorithms, local search procedures based on metrics, etc.). Such approaches mostly disregard the a priori knowledge about the problem that can have a significant impact on the algorithm efficiency. A systematic approach to the development of an optimization method, based entirely on the characteristics of the problem under consideration, can result in an algorithm whose efficiency significantly outperforms conventional approaches.

In addition, this approach could explore theoretical relations between similar optimization problems and utilize known efficient methods for one problem to address subproblems of the other. For example, scheduling independent jobs on identical machines, known as $P||C_{max}$ in the three-filed notation [5], Bin Packing Problem (*BPP*) and Multiple Subset Sum Problem (*MSSP*) [11] are $\mathcal{NP}$ problems that share a common theoretical background and can often be considered as dual problems [8]. The correlation between these problems should be explored in both theoretical considerations and the development of efficient solution methods. Determination of lower bounds is a very good example of leveraging this correlation [2, 8].

Our main contribution is the development of Decomposition-based Iterative Stochastic Transformation (*DIST*), an efficient heuristic algorithm for $P||C_{max}$, that exploits correlation between $P||C_{max}$, *BPP*, and *MSSP*. Their mutual theoretical properties can improve performance of the proposed algorithm and guarantee the optimality of the obtained solution given enough time. Our approach employs a partitioning technique that attempts to iteratively improve the current $P||C_{max}$ solution by solving its subproblems as the instances of *MSSP*. It starts with random partitions

defining smaller subproblems as they have higher potential for improving the current solution. If the current solution is not improved algorithm selects larger subproblems although potential to improve the current solution steadily decreases. Thus, to not waste time and resources, the individual steps of the algorithm are time limited. Nevertheless, *DIST* can provide high quality solutions within a short execution time, as it is demonstrated by the experimental evaluation and comparison with the state-of-the-art algorithm [4] on the same benchmark sets of instances.

The paper is organized in the following way. After a brief introduction, the MIP formulation of all considered problems is provided and their analogy is discussed in Section 2. The proposed algorithm is described in Section 3. Section 4 contains an experimental evaluation on the benchmark instances from the relevant literature. Concluding remarks and directions for future research are provided in Section 5.

2. FORMULATION OF RELATED OPTIMIZATION PROBLEMS

In this section we describe *MSSP*, $P||C_{max}$, and *BPP* and provide their MIP formulations.

MSSP can be described as follows. Let $W = \{1, \dots, g\}$ be a given set of $g \in N$ items, with positive integer weights $w = \{w_1, \dots, w_g\}$, and let $K = \{1, \dots, k\}$ represents a set of $k \in N$ identical knapsacks with capacity $c \in N$. The objective is to select a subset of items that can be packed in the knapsacks in such a way to maximize the total weight of selected items.

For a given 4-tuple (W, w, K, c), mathematical programming formulation of *MSSP* is given by the Integer Linear Program (ILP) in [1]:

$$\max \sum_{i \in K} \sum_{j \in W} w_j x_{ij} \tag{1}$$

$$\text{s.t.} \quad \sum_{i \in \mathrm{K}} x_{ij} \leq 1, \qquad j \in W, \tag{2}$$

$$\sum_{j \in W} w_j x_{ij} \leq c, \qquad i \in \mathrm{K}, \tag{3}$$

$$x_{ij} \in \{0,1\}, \qquad j \in W, i \in \mathrm{K}. \tag{4}$$

where x_{ij} takes value 1, if item j is assigned to knapsack i, and 0 otherwise.

MSSP is a specialization of Multiple 0-1 Knapsack Problem and generalization of Subset Sum Problem (*SSP*) [11].

When describing $P||C_{max}$ scheduling problem [9], a set $M = \{1, \dots, m\}$ of $m \in N$ identical independent machines, and a set $J = \{1, \dots, n\}$ of $n \in N$ independent jobs with processing times $p = \{p_1, \dots, p_n\}$ should be given. Each job should be assigned to exactly one machine in such a way to minimize the latest machine completion time (makespan) denoted by C_{max}. Let us define C_i as the sum of processing times of jobs assigned to machine i. Then $C_{max} = \max\{C_1, \dots, C_i, \dots, C_m\}$.

According to [12], $P||C_{max}$ for a given 3-tuple (J, p, M) can be formulated as the following ILP:

$$\min C_{max} \tag{5}$$

$$\text{s.t.} \quad \sum_{i \in \mathrm{M}} x_{ij} = 1, \qquad j \in \mathrm{J}, \tag{6}$$

$$\sum_{j \in \mathrm{J}} p_j x_{ij} \leq C_{max}, \qquad i \in \mathrm{M}, \tag{7}$$

$$x_{ij} \in \{0,1\}, \qquad i \in \mathrm{M}, j \in \mathrm{J}. \tag{8}$$

where x_{ij} takes value 1, if job j is assigned to machine i, and 0 otherwise.

Theorem 1: $P||C_{max}$ for a given 3-tuple (J, p, M) is equivalent to *MSSP* for 4-tuple (J, p, M, c) in which all jobs are used, and c is minimal, i.e.:

$$\min c \tag{9}$$

$$\text{s.t.} \quad MSSP \quad \text{for} \quad (J, p, M, c), \tag{10}$$

$$\sum_{i \in M} x_{ij} = 1, \qquad j \in J, \tag{11}$$

$$x_{ij} \in \{0,1\}, \qquad j \in J,\ i \in M. \tag{12}$$

Proof 1: Maximization required in (10), if feasible, under the constraints (11) will always return the sum of all items, regardless the value of c. Constraints (11) are required to strengthen constraints (2) and to ensure the constraints (6) are satisfied. Having included all items (jobs) in the selection, as it is stated by constraints (11), constraints (3) reduce to constraints (7). After these reductions, c can be observed as C_{max}, and its minimization in (9) leads to the solution of $P||C_{max}$. ■

Next, we consider *BPP*, provide its description and MIP formulation. For a given bins capacity q, and a set $X = \{1, \dots, x\}$ of $x \in N$ items with positive weights $y = \{y_1, \dots, y_x\}$, the goal of *BPP* is to determine minimal number of bins $b \in N$ such that each item can be assigned to exactly one bin from the set of bins $B = \{1, \dots, b\}$.

BPP can be considered as dual problem to $P||C_{max}$ [8] and can also be interpreted via *MSSP*, providing suitable basis for our consideration.

ILP for *BPP* described by 3-tuple (X, y, q) is formulated in [12]:

$$\min |B| \tag{13}$$

$$\text{s.t.} \quad \sum_{i \in B} x_{ij} = 1, \qquad j \in X, \tag{14}$$

$$\sum_{j \in X} y_j x_{ij} \leq q, \qquad i \in B, \tag{15}$$

$$x_{ij} \in \{0,1\}, \qquad i \in B,\ j \in X. \tag{16}$$

where x_{ij} takes value 1, if item j is assigned to bin i, and 0 otherwise.

Theorem 2: *BPP* for a given 3-tuple (X, y, q) is equivalent to *MSSP* described by 4-tuple (X, y, B, q) where all jobs are used, and $|B|$ is minimal, i.e.,

$$\min |B| \tag{17}$$

$$\text{s.t.} \quad MSSP \quad \text{for} \quad (X, y, B, q), \tag{18}$$

$$\sum_{i \in B} x_{ij} = 1, \qquad j \in X, \tag{19}$$

$$x_{ij} \in \{0,1\}, \qquad j \in X,\ i \in B. \tag{20}$$

Proof 2: Like in Proof 1, maximization required in (18), if feasible, under the constraints (19) will always return the sum of all items, regardless the value of q. Constraints (19) are required to strengthen constraints (2) and to ensure that constraints (14) are satisfied. Having included all jobs in the selection, as it is stated by constraints (19), constraints (3) reduce to constraints (15). After these transformations c can be observed as q, and its minimization in (17) provides a solution for *BPP*. ■

Based on the provided relationship between the considered problems, we can formulate our *DIST* algorithm.

3. DIST DESCRIPTION

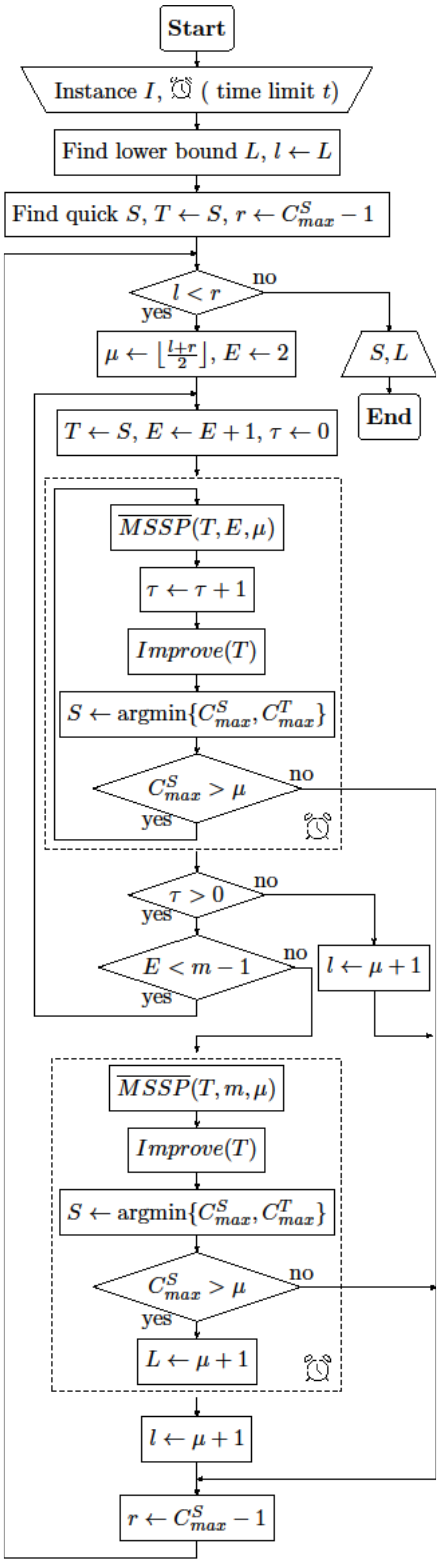

Figure 1: DIST block diagram

The basic idea of *DIST* is the divide and conquer strategy, i.e., solving a part of the problem with an aim to improve the quality of the current solution. More precisely, the set of machines is partitioned in two parts and the one containing the most heavily loaded machine is considered and solved as an instance of *MSSP*. The only way to improve the current solution is to decrease the load of the most heavily loaded machine. In our experience, iteratively applying this procedure can lead to a high-quality solution in a short amount of time. Figure 1 shows the block diagram of *DIST*. The *DIST* algorithm accepts the instance I of the problem and the time limit for the timed parts. Time limit is enforced within the operating system, and it can interrupt any action in the algorithm. In Figure 1, the time limited parts are represented with a rectangle containing a clock in the lower right corner. Main components of *DIST* algorithm are explained in the reminder of this section.

3.1. Lower bound strategies

The first very important step for performance, and the primary strategy for the solver's ability to guarantee the optimality of the solution, is to find the best possible lower bound. Given an instance of $P||C_{max}$, lower bound L is obtained as a maximum of several lower bounds from the literature. With the assumption that jobs are sorted by processing time such that $p_1 \geq p_2 \geq \cdots \geq p_n$ trivial lower bound is defined as $L_{TV} = \max\{L_0, L_1, L_2, L_\nu\}$, where $L_0 = \left\lceil \frac{1}{m}\sum_{j=1}^{n} p_j \right\rceil$, $L_1 = p_1$, $L_2 = p_m + p_{m+1}$, and $L_\nu = \sum_{j=n-\nu+1}^{n} p_j$, where $\nu = \lceil n/m \rceil$ [2]. L_{DM} [2], and L_{HS} [8] are based on *BPP*. More complex one is L_θ [2]. Every lower bound is improved using lifting procedure [6, 7], so we get $\widehat{L_{TV}}$, $\widehat{L_{DM}}$, $\widehat{L_{HS}}$, and $\widehat{L_\theta}$. Finally, $L = \max\{\widehat{L_{TV}}, \widehat{L_{DM}}, \widehat{L_{HS}}, \widehat{L_\theta}\}$.

3.2. Upper bound strategies

The next step (*Find quick S* in Figure 1) considers finding initial solution, i.e., upper bound. It is important to provide our algorithm with a reasonably good solution quickly, thus, we apply the well-known longest-processing-time-first scheduling (LPT). The principle is simple: as long as there are unscheduled jobs, take the longest among them and assign to the least loaded machine - earliest start (ES) scheduling rule. Approximation ratio of LPT is $\frac{4}{3} - \frac{1}{3m}$ [5].

In an improvement of LPT, the $SLACK$ heuristic [3], the array of initially sorted jobs is divided into m tuples of approximately $\frac{n}{m}$ size. The tuples are sorted in non-increasing order according to the difference between the longest and the shortest job and concatenated into a joint list of jobs that are scheduled according to ES scheduling rule. Consequently, approximation ratio of $SLACK$ is improved to $\frac{4}{3} - \frac{1}{3(m-1)}$ [3]. Finally, $S = \text{argmin}\{C_{max}^{LPT}, C_{max}^{SLACK}\}$.

3.3. Heuristic strategy

The main part of the algorithm is a binary search for a solution with the minimum makespan in the interval $[l, r]$, initially set to $[L, C_{max}^{S} - 1]$, whose size is adjusted accordingly during the search. The goal is to find a solution with makespan less than or equal to the middle value μ (see Figure 1). For each value μ, the search is performed by timed iterative trials. Temporary solution (T) is set to S, E is incremented until $E = m - 1$, and the transformation counter (τ) is initialized. The goal is to improve T by using an exact *MSSP* solver [15] on a subproblem containing the most heavily loaded machine, $E - 1$ randomly selected machines, and all jobs on them. Addressing $P||C_{max}$ by *SSP* [7, 14] and *MSSP* [6] was limited to 2-machines subproblems. To the best of our knowledge, we are the first to include more than two machines to create subproblems. After each transformation, *Improve* procedure is applied to T and S is updated. *Improve* procedure is using MSSP for $E = 2$ where the second machine is chosen in sequence from all other machines and that procedure is repeated until there is an improvement. If $C_{max}^{S} \leq \mu$, the search interval is adjusted and algorithm proceeds to the next step. If the time limit is exceeded without at least one transformation for the given E, the search for μ is unsuccessful and the left part of the interval must be discarded. If

no solution is found for $E = m - 1$ but some transformations are performed, then there is a chance to obtain an exact solution by solving the *MSSP* for all jobs and machines within a given time limit. If the transformation is performed and a feasible solution is not found, a new lower bound can be guaranteed.

4. EXPERIMENTAL EVALUATION

We evaluate *DIST* in comparison with state-of-the-art solver Improved Arc-Flow (*IAF*) [4]. Comparison is done on 700 instances with ratio $\frac{n}{m} = 2$ [13]. Our solver is tested using GCC version 10.4.0, Linux 4.15.0-143-generic, Ubuntu 18.04.5 LTS on Intel(R) Core(TM) i5-6400 CPU @ 2.70GHz with 8GB RAM. IAF solver is tested by IAF authors using CPLEX 12.10 on Intel(R) Core(TM) i7-4930K CPU @3.40 GHz and 34.0 GB of RAM. Obtained results enable us to ignore the superiority of hardware resources used to execute IAF.

Table 1: DIST and IAF performance comparison on used benchmark instances

Solver	Performance						
	Time [s]	**Time to Best [s]**	**Trans.**	**Trans. to Best**	**Optimality Guaranties [%]**	**Best [%]**	**Number of Tests**
IAF (state of the art)	184.78	184.78	/	/	100	100	1
DIST limit 0.06s	107.48	***32.56***	1069608	440121	78.71	96.64	30

Having in mind that *DIST* is a stochastic heuristic, it is run 30 times for all instances to obtain statistical significance. Obtained results are provided in Table 1. The second and the third columns show the sum of the total execution times and sum of times required to obtain the best solution, respectively, for all 700 instances. Like for times, sum of the total number of performed transformations and the sum of number of transformations until the best solution is found are presented as Trans. and Trans. to Best. The percentage of instances for which algorithms can guarantee optimality of solutions is shown in the next column. The penultimate column shows the percentage of best solutions in all runs (700*30), while the last column shows the number of runs per instance. For DIST we summed up the worst times and the worst numbers of transformations out of the all the best provided solutions in 30 repetitions for each of 700 instances.

Both algorithms have solved all instances. Although it is not an exact solver, our heuristic can guarantee the optimality of the solution in almost 79% of cases. As *DIST* achieved almost 97% optimal solutions, it can be declared reliable heuristic in this experiment. *DIST* execution time is strongly related to given time limits. For all tested instances, 0.06s time limit was enough to obtain optimal solution. In comparison with *IAF*, *DIST* can find solutions about 6 times faster, and complete search 40% faster. Most instances require significantly less than 0.06s for subproblem.

5. CONCLUSION

We considered a systematic approach to the development of an optimization method based on the characteristics of the problem, exploring theoretical relations between similar optimization problems, and utilizing known efficient methods for one problem to address subproblems of the other. Our approach has been explained on the problem of scheduling independent jobs on identical machines ($P||C_{max}$) as a case study. The resulting efficient heuristic algorithm, named Decomposition-based Iterative Stochastic Transformation (*DIST*), explores the analogy between $P||C_{max}$, Bin Packing Problem (*BPP*) and Multiple Subset Sum Problem (*MSSP*). We compared *DIST* to state-of-the-art exact algorithm on benchmark instances from literature. As expected, *DIST*

performed better w.r.t. total runtime while reaching the optimal solution for all instances. However, even though *DIST* is heuristic algorithm, it guarantees optimality in almost 79% benchmark instances.

As future work we plan to improve *DIST* performance by including new search mechanisms and lower bounds and exploring more benchmark instances. As explored problems can be easily transformed one into another, the same strategy can be used to develop efficient algorithms for any of them and that could be another avenue of future work. Along the same lines, the proposed approach could be applied to other sets of similar optimization problems.

ACKNOWLEDGEMENT

This work was supported by the Ministry of Science, Technological Development and Innovations of Republic of Serbia, agreements nos. 451-03-47/2023-01/200029 and 451-03-47/2023-01/ 200122, Science Fund of Republic of Serbia under the project Advanced Artificial Intelligence Techniques For Analysis And Design Of System Components Based On Trustworthy Blockchain Technology (AI4TrustBC), and Penn State Great Valley Big Data lab. We thank Gharbi, Anis. and Bamatraf, Khaled for providing IAF results.

REFERENCES

[1] Caprara, A., *et al.* (2000). The multiple subset sum problem. SIAM Journal on Optimization, 11(2), 308-319.

[2] Dell'Amico, M., & Martello, S. (1995). Optimal scheduling of tasks on identical parallel processors. ORSA Journal on Computing, 7(2), 191-200.

[3] Della Croce, F., & Scatamacchia, R. (2020). The longest processing time rule for identical parallel machines revisited. Journal of Scheduling, 23(2), 163-176.

[4] Gharbi, A., & Bamatraf, K. (2022). An Improved Arc Flow Model with Enhanced Bounds for Minimizing the Makespan in Identical Parallel Machine Scheduling. Processes, 10(11), 2293.

[5] Graham, R. L. (1969). Bounds on multiprocessing timing anomalies. SIAM Journal on Applied Mathematics, 17(2), 416–429

[6] Haouari, M., & Jemmali, M. (2008). Tight bounds for the identical parallel machine-scheduling problem: Part II. International Transactions in Operational Research, 15(1), 19-34.

[7] Haouari, M., *et al.* (2006). Tight bounds for the identical parallel machine scheduling problem. International Transactions in Operational Research, 13(6), 529-548.

[8] Hochbaum, D. S., & Shmoys, D. B. (1987). Using dual approximation algorithms for scheduling problems theoretical and practical results. Journal of the ACM, 34(1), 144-162.

[9] Lawrinenko, A. (2017). Identical parallel machine scheduling problems: structural patterns, bounding techniques and solution procedures (Doctoral dissertation, Friedrich-Schiller-Universität Jena).

[10] Maleš, U., *et al.* (2023). Controlling the Difficulty of Combinatorial Optimization Problems for Fair Proof-of-Useful-Work-Based Blockchain Consensus Protocol. Symmetry, 15(1), 140.

[11] Martello, S., & Toth, P. (1990). Knapsack problems: algorithms and computer implementations. John Wiley & Sons, Inc.

[12] Mokotoff, E. (2004). An exact algorithm for the identical parallel machine scheduling problem. European Journal of Operational Research, 152(3), 758-769.

[13] Mrad, M., & Souayah, N. (2018). An arc-flow model for the makespan minimization problem on identical parallel machines. IEEE Access, 6, 5300-5307.

[14] Ostojić, D., *et al.* (2022). Comparative Analysis of Heuristic Approaches to P|| Cmax, Proc. 11th International Conference on Operations Research and Enterprise Systems, ICORES 2022, (virtual), Feb. 3-5, 2022, pp. 259-266.

[15] Pisinger, D., & Toth, P. (1998). Knapsack problems. Handbook of Combinatorial Optimization: Volume1–3, 299-428.