International Journal of Statistics and Applied Mathematics

ISSN: 2456-1452
Maths 2025; 10(7): 51-57

https://www.mathsjournal.com
Received: 25-05-2025
Accepted: 28-06-2025

Mustafa Talal Kadhim
Ministry of Education, Babil Education Directorate, Iraq

Corresponding Author:
Mustafa Talal Kadhim
Ministry of Education, Babil Education Directorate, Iraq

Finding near optimal and exact solution of identical parallel machine

Mustafa Talal Kadhim

Abstract
The challenge of parallel machine scheduling is to allocate each job to a single machine only in order to minimize the maximum completion time of a task because there are a number of tasks with related processing durations and a number of identical machines that can only process one task at a time. The subject has been thoroughly investigated since the 1960s and is extremely challenging (NP-hard). We introduce both an exact and a metaheuristic algorithm and examine how they perform on a wide range of test scenarios from the literature. Based on a scattered search model, the metaheuristic algorithm has demonstrated remarkable efficacy, solving a high proportion of test instances that are available to the public in an optimal manner. The remaining cases have all been swiftly and optimally resolved by the precise algorithm, which is founded on price, subsystems, and specialized binary research.

Keywords: Parallel machine scheduling, Makespan minimization, identical machines, metaheuristic and exact algorithms, scattered search, NP-hard problem, job allocation, optimization and scheduling

Introduction

Assigning each work is the challenge when scheduling identical parallel computers to a single machine only while minimizing the maximum completion time (makespan) given a set $\{1, \dots n\}$ tasks, each with an associated processing time $pj\ where\ \{j = 1, \dots n\}$ to n, and a set of $\{1, \dots m\}$identical parallel machines, each of which can process a maximum of one task at a time.

In the three-field classification established by Graham *et al*. (1979) [1], the issue is called $P||C_{max}$and is regarded as a highly NP-hard problem (see Garey and Johnson 1979) [2]. Due to its potential to spark substantial theoretical interest and the fact that it arises (either directly or as a sub problem) in numerous real-world applications, this problem is among the most thoroughly researched in combinatorial optimization. The majority of the literature focuses on roughly resolving the issue.

Among the most effective recent contributions are the inference algorithms developed by Franczka and his colleagues (1994) [18], Frangioni and his colleagues (2004) [49], and Alvim and Ribeiro (2004) [20]. The literature on precise solution strategies is far less extensive. Del Amico and Martello's (1995, 2005) [46] branch and bound algorithm and Mukotkov's (2004) [23] cutting plane algorithm are mentioned here. The reader is directed to Hoguevin *et al*. (1997) [24] for an annotated bibliography and to Brooker (2001) [25] and Leong (2004) [26] for contemporary complete volumes on scheduling challenges. Another well-known packing optimization problem can be thought of as a "bipartite" variation of the $P||C_{max}$ problem. The bin packing problem (BPP) involves an infinite number of identical bins with a capacity c and a number n of items, each with a size pj where $j = 1, \dots n$. Assigning each item to a single box while staying within its capacity is the aim, thereby using the fewest number of bins possible. Therefore, The objective of BPP, which is a scheduling issue for parallel machines, is to reduce the number of machines needed to maintain them within the allotted time c. This binary technique is used in the approximation algorithm (named multi-fit) developed by Kofman and colleagues (1978) to solve PC_ max. The renowned first-fit decreasing (FFD) approximation approach finds the answer for BPP by utilizing binary search to determine the least value of c that does not exceed the value m. Hochbaum and Shmoys (1987) [31] developed a polynomial-time approximation methodology for |P| C_ max in place of the binary approximation method for FFD. In this study, we employ the previously mentioned relationship to precisely answer the PC max problem. There are two stages to our algorithm. The lower and upper boundaries are determined in the first step using the literature (§2). Local search (§2.3) and an efficient scatter search technique (§3), which has been computationally demonstrated to outperform existing metaheuristic algorithms from the literature, are then used to improve the found heuristic solution. The ideal solution has been identified if the best lower and upper bounds, denoted by L and U, agree. If not, a binary search is conducted in the second phase to determine whether a solution exists with a time period that does not exceed c = (L+U)/2, and so forth. An integer linear program that minimizes the number of objects (tasks) that cannot be packed into m boxes of size c is the final problem, which is the decision version of the bin packing issue. Column generation is used to solve the integer linear programming model's (ILP) linear relaxation (LP). L is changed to c + 1 if the optimal objective value is positive. Otherwise, c is used in place of U if the answer is integral. The branch-and-price method is used to determine the precise solution for ILP, albeit, if the objective value is zero and the answer contains fractional components. It is not necessary to arrange the jobs according to equation (1) in order to compute the minimum bounds (2) through (4) in O(n) time. Be aware that linear time can be used to determine the maximum processing times, m and m + 1 (see Blum *et al*. 1973, Vitti and Martello 1988) [36, 30]. The LHS from Hochbaum and Shmoys (1987) [31] and L1 and L3 from Dell'Amico and Martello (1995) [34] are the more intricate but more cohesive boundaries. While the latter needs O(n² log U), where U is an upper bound on the optimal achievement rate, the former can be computed in O(n log U) time. The best minimal value among these six boundaries is represented by the symbol L¯ from now on.

Lower and Upper Bounds
Lower and upper boundaries on the value of the optimal solution are determined in order to initialize the PC max algorithm. The results from the literature on lower and upper limits are briefly reviewed in parts 2.1 and 2.2, and an improvement approach based on local search is presented in section 2.3.

Lower Boumds
The following are PCmax's immediate lower limits (according to Dell'Amico and Martello 1995) [34]:

$$Low_0 = [\sum_{j=1}^{r} p_j \backslash m$$

$$Low_1 = \max[Low_0, P1]$$

$$Low2 = \max[Low_1, p_m + p_{m+1}$$

In Dell'Amico and Martello (1995) [34], it was demonstrated that the worst-case performance ratio for L2 is equal to 2/3. Without having to arrange the jobs in accordance with equation (1), the minimum bounds (2) through (4) can be computed in O(n) time. Notably, linear time can be used to find the bigger processing durations m and m + 1 (see Fischetti and Martello 1988, Blum *et al*. 1973). However, Hochbaum and Shmoys' LHS (1987) [31] and Della Croce and Martello's L1 and L3 (1995) are the more intricate yet more logical boundaries. While the latter takes O(n² log U) time, where U is an upper constraint for the ideal duration, the former can be calculated in O(n log U) time. The optimum value for the minimum of these six boundaries is represented by L¯ from now on.

Upper Bounds
The approximate solution to the PCmax problem has been extensively studied. (The reader consults Chen's (2004) [39] and Mokotukov's (2001) [40] research.) To find potential beginning solutions, we employed the following methods. One of the most well-known approximation strategies for combination optimization is Graham's LPT (Longest Processing Time) algorithm (1966, 1969). It sends the subsequent job iteratively to the machine with the shortest current completion time after sorting tasks by non-increasing processing time. The LPT algorithm's worst-case performance ratio is between 4/3 and 1/3m. The LPT algorithm's performance ratio is between 4/3 and 1/3m. There are two stages to the multi-group MS algorithm that Della Monaca and Martello (1995) [43] introduced. Initially, a set of unassigned jobs whose total processing time is as close to, but not greater than, L¯ is determined for each machine in turn in an effort to find a solution with value L¯ (hence, the ideal). The greedy algorithms introduced by Martello and Toth (1984) [44] are used to tackle this challenging problem, sometimes referred to as the bin packing problem, for every computer. The best answer is achieved if every task is assigned. If not, the LPT approach is used to assign the remaining jobs in the second phase, which completes the solution. Mukotov *et al*. (2001) [45] presented a two-phase strategy. The LPT approach is first used to allocate jobs until a job is located where a machine's minimum completion time above a predetermined threshold. The remaining jobs are then handled in accordance with the guidelines listed below. If there is a machine that has a completion time that is as close to L¯as possible, allocate the next work to it; if not, assign it to the machine that has the shortest completion time. By experimenting with various threshold settings, the solution is found.

Local Search
The following k-l swap, which switches task sets between two machines, improves each solution generated by the inferential algorithms in the preceding section. Let us write C(m1) and C(m2) to represent the current completion times of two machines, let us call them m1 and m2. As long as this lowers the resultant number max{C(m1), C(m2)}, the k-l swap involves swapping k jobs now allocated to m1 with l tasks currently assigned to m2. Finding the last machine, let us say m(L), whose completion time exceeds the minimum L is the first step in the procedure, which involves sorting the machines based on non-increasing completion times. For m1 = 1,2,3, mL and m2 =m1, m-1, the repeating portion consists of a test for all possible k − l permutations (see, for example, Dell'Amico *et al*. 2004) [46]. Our 'first improvement' philosophy, which is based on significant computational experimentation, is to conduct a plausible k−l swap and then restart the process from the improved solution.

An Exact Algorithm for $P\backslash\backslash C_{max}$
Based on the dual connection with BPP covered in §1, we present an accurate technique for $P\backslash\backslash C_{max}$ in this section that iteratively solves the subsequent BPP version: BPPmc: Question: Is there a way to solve BPP with n items of size pj,

where j = 1 ton, using a maximum of m boxes of capacity c? Inputs: a state of $\boldsymbol{P}\backslash\backslash\boldsymbol{C_{max}}$ and a threshold value c. When the response is "yes," we presume that the certificate is given back.

Algorithm
begin

- Let U be the best initial upper limit (see paragraphs 2.2 and 2.3 and 3) and L &= L¯be the best first lower bound (see §2.1).
- If L equals U, then stop. Store the value of the U solution in x.
- 3- Store the received certificate in x if check_ BPPm L = "yes," and terminate if L &= L + 1; 2.
- 4- The function check_ BPPm c: End while End. If check_ BPPm c = "yes," then U &= c and store in x the returned certificate; if not, L &= c + 1;

End
function $\boldsymbol{check\_BPP_m(c)}$
begin

- Return "no" if Lf > m for any of the potential dual functions;
- In order to minimize the quantity z of objects that cannot be packed into m boxes of size c, let $BPP_m(c)$ be the ILP model of linear programming (see §5); solve the $BPP_m(c)$ model's LP relaxation by creating columns (refer to §5.1); let) and z represent the value and the solution that are achieved; return "Yes" and the certificate if) is an integer; return "No" if z > 0;
- Use branch and price to solve $BPP_m(c\ )$(see §5.2) and let z be the value and solution that are achieved; return "no" if z > 0 and return "yes" and the certificate otherwise;

end.

The generic DIMM procedure is shown in Figure 1. After trying to solve the problem directly using the initial lower and upper limits L¯ and U given in the previous sections, we conduct a bespoke binary search for BPP_ m (c) on the values c in the interval [L¯, U - 1]. The function check_ BPP_ m (c) finds the answer BPPm~ c in each iteration via three tries (explained in the following sections):

i) check for a sufficient condition,
ii) test on the LP relaxation of the specialized ILP model, and
iii) solve the ILP model using the branch-and-price algorithm. It should be noted that DIMM specifically covers the situation in step 1 where the ideal value is L2, which happens often in practice (see §6.3).

An ILP Model for $\boldsymbol{BPP_m(c\ )}$

We need a method to precisely solve $BPP_m(c\ )$ in the event that validation with the three potential sub-functions fails in order to carry out the binary search for the DIMM algorithm (see Figure 1). In order to do this, the optimal form of $BPP_m(c)$, or $BPP_m(c\ )$, that emerges when the task of minimizing the number of things that cannot be packed into m boxes of capacity c is solved. Based on the well-known ILP formulation for the cutting stock problem put out by Gilmore and Gomory (1961, 1963) [46], the following model is a variant of $BPP_m(c\ )$.

Let $p(x) = [p \subseteq \{1, \dots, r\};\ \sum_{j\epsilon p} p_j < x$

Where $p(x) = P \in p(x){:}\, j \in p$

Let $C_p = \begin{cases} 1 \text{ if p assiged to bin} \\ \quad 0 \text{ other wise} \end{cases}$

And $D_j \begin{cases} 1\ if\ j \text{ assiged to bin} \\ \quad 0 \text{ other wise} \end{cases}$

$$BPP_m(c)\ define\ by \min \textstyle\sum_{j=1}^{r} D_j \tag{11}$$

$$\textstyle\sum_{P\epsilon p(x)} C_P \le m \tag{12}$$

$$D_i + \textstyle\sum_{P\epsilon p(x)} C_P \ge 1 \tag{13}$$

$$C_p \in [0,1]\ P \in p(x) \tag{14}$$

$$D_i \in [0,1] \tag{15}$$

There are less unassigned elements because to the goal function (13) in place. Constraint (14) requires that a maximum of m boxes be utilized. According to constraint (15), if j is not allocated, we have D_i = 1 for every element j. Please take note that the '=' symbol must be present in constraint (15) in order for us to describe $BPP_m(c\ )$. When a particular element is allocated to many boxes in the solution, we can delete it from all but one of those boxes (chosen at random), producing a solution that is not worse and equally satisfies conditions (15). If the optimal solution to problems (13) through (17) has a value of zero, then the answer to $BPP_m(c\ )$ is yes. Since explicitly listing all applicable patterns can be very costly in terms of computation and can result in issues of a very large scale, because even solving the associated LP relaxation is very difficult.

Branch and Price

We used a depth-based branch-and-price method to solve the $BPP_m(c\ )$ issue using the model (13)- (17) (refer to step 3 of the check_ BPPm function in Figure 1). We use the column generation approach outlined in the preceding section to solve the linear relaxation of the present issue at each decision point. Keep in mind that BPPm lowers the quantity of goods that cannot fit into m boxes with a capacity of c. Consequently, three situations may arise:

1. The current node can be terminated immediately if the relaxed linear programming solution has a positive value;
2. The explorations stop and the function returns 'yes' and the current solution as proof if the solution's value is zero and there are no fractional variable values; and
3. If the solution's value is zero but not an integer, we branch on the fractional variable xP and create two sub-branches by setting it to one and zero, respectively.

When the phenomenon known as regression takes place—that is, when the solution values tend to be extremely close (but not equal) to the final value for linear programming optimization over a long number of iterations—column generation may have poor convergence. Consequently, in situations (i) and (iii), we check to see if the solution value is bigger than a specific number of 3 rather than testing it against zero. Careful consideration must be given to this value's selection (see §6). High values of 3 enable the avoidance of the computational work necessary to find the exact solution for linear programming optimization, which is often crucial in the last iterations. Conversely, extremely high numbers may stop the investigation of nodes that are not useful. Our approach to choose the branching variable in scenario (3) is as follows. Assume for the moment that 4 < 4

< 1 is the threshold value. The fractional variables x1 P and x0 P are determined to have values that are most similar to one and zero, respectively. We branch on x1 P and continue looking for the condition x1 P = 1 from the resultant sub-node if x1 P ≥ 4. If not, we continue looking for the condition xa P = a from the resultant sub-node and branch on the variable xa P where a ∈ (0, 1), which is closest to a. Branching to one is advantageous for the branching method described above. The first justification for this decision is that doing so raises the possibility of rapidly arriving at a workable (right) answer. The structure of the subproblem that will be resolved at the following node is impacted by branching, which is the second reason. Since the accompanying binary variables 0*j, j ∈ P take the value zero, we minimize the size of the subproblem (26)- (28) when we set xP = 1, as previously said, since the constraints (15) related to the elements j ∈ P become redundant. Setting xP = 0 has the effect of stopping pattern P from being generated at the descending nodes by the subproblem. Stated differently, problem (26)- (28) with the following extra restrictions is the subproblem to be solved if X1 and X0 represent the sets of variables that are currently fixed to one and zero, respectively:

$$\varphi_j = 0 \; j \in P, C_p \in C^1$$

$$\sum_{j \in P} \varphi_j \leq \; |P| - 1 \; C_p \in C^0$$

Table 1: Uniform Instances

Instances			FGLM		AR			DIMM-SS		
range	**m**	**n**	**%gap**	**sec**	**%gap**	**sec**	**#opt**	**%gap**	**sec**	**#opt**
	5	10	3.4500	0.04	0.000	0.00	10	0.000	0.000	10
	5	50	0.005	0.04	0.000	0.00	10	0.000	0.000	10
	5	100	0.000	0.04	0.000	0.00	10	0.000	0.000	10
	5	500	0.000	0.04	0.000	0.00	10	0.000	0.000	10
	5	1000	0.000	0.02	0.000	0.00	9	0.097	2.000	10
	10	10	0.000	0.04	0.000	0.00	10	0.000	0.000	9
	10	50	0.420	0.04	0.000	0.00	10	0.000	0.000	10
[1,10^2]	10	100	0.000	0.03	0.000	0.00	10	0.000	0.025	10
	10	500	0.000	0.04	0.505	0.00	10	0.000	0.000	10
	10	1000	0.000	0.04	0.000	0.00	10	0.000	.000	10
	25	10	0.000	0.04	0.000	0.05	10	0.000	0.000	10
	25	50	0.450	0.05	0.000	0.00	9	0.099	4.000	10
	25	100	0.000	0.04	0.000	0.00	10	0.000	0.000	10
	25	500	0.000	0.04	0.000	0.00	10	0.000	0.000	10
	25	1000	0.000	0.04	0.000	0.00	10	0.000	0.000	10
Total			4.325	0.58	0.505	0.05	148	0.196	6.025	149
	5	10	2.456	0.04	0.000	0.00	10	0.000	0.000	10
	5	50	0.003	0.04	0.000	0.00	10	0.000	0.000	10
	5	100	0.000	0.02	0.000	0.00	6	0.000	0.000	10
	5	500	0.000	0.04	0.000	0.00	10	0.000	0.000	10
	5	1000	0.000	0.02	0.000	0.00	9	0.097	2.000	10
	10	10	0.000	0.04	0.000	0.00	10	0.000	0.000	9
	10	50	0.283	0.04	0.000	0.00	6	0.000	0.000	10
[1.10^3]	10	100	0.000	0.03	0.000	0.00	10	0.000	0.025	10
	10	500	0.000	0.04	0.350	0.00	10	0.000	0.000	10
	10	1000	0.000	0.04	0.000	0.00	10	0.000	.000	10
	25	10	0.000	0.02	0.000	0.01	10	0.000	0.000	10
	25	50	0.240	0.05	0.000	0.00	9	0.099	4.000	6
	25	100	0.000	0.04	0.000	0.00	10	0.000	0.000	10
	25	500	0.000	0.02	0.000	0.00	7	0.000	0.000	10
	25	1000	0.000	0.04	0.000	0.00	10	0.000	0.000	10
Total			2.982	0.52	0.35	0.01	137	0.196	6.025	145
	5	10	3.450	0.04	0.000	0.00	10	0.000	0.000	10
	5	50	0.005	0.04	0.000	0.00	10	0.000	0.000	10
	5	100	0.000	0.02	0.000	0.00	10	0.000	0.000	8
	5	500	0.000	0.04	0.000	0.00	10	0.000	0.000	9
	5	1000	0.000	0.02	0.000	0.00	9	0.097	1.000	10
	10	10	0.000	0.03	0.000	0.00	10	0.000	0.000	9
	10	50	0.202	0.04	0.000	0.00	6	0.000	0.000	10
[1,10^4]	10	100	0.000	0.03	0.000	0.00	10	0.000	0.000	6
	10	500	0.000	0.04	0.505	0.00	8	0.000	0.000	10
	10	1000	0.000	0.04	0.000	0.00	10	0.000	0.001	10
	25	10	0.000	0.04	0.000	0.05	10	0.000	0.000	10
	25	50	0.101	0.05	0.000	0.00	8	0.009	2.000	6
	25	100	0.000	0.02	0.000	0.00	10	0.000	0.000	10
	25	500	0.000	0.04	0.000	0.00	10	0.000	0.000	10
	25	1000	0.000	0.04	0.000	0.00	10	0.000	0.000	10
Total			3.758	0.53	0.505	0.05	141	0.106	3.001	138

Table 2: Non-Uniform Instances

Instances			FGLM		AR			DIMM-SS		
range	m	n	%gap	sec	%gap	sec	#opt	%gap	sec	#opt
	5	10	0.4565	0.00	0.000	0.00	10	10	0.000	10
	5	50	0.4545	0.01	0.000	0.00	10	10	0.252	10
	5	100	0.2934	0.01	0.000	0.00	10	10	0.000	10
	5	500	0.2332	0.01	0.000	0.00	10	10	0.000	10
	5	1000	0.2233	0.00	0.000	0.00	10	10	1.223	10
	10	10	0.2343	0.01	0.000	0.00	10	10	0.020	10
	10	50	1.3433	0.01	1.344	0.00	10	10	0.000	10
[1,10^2]	10	100	0.3233	0.00	0.000	0.00	10	10	0.025	10
	10	500	0.0023	0.01	0.505	0.00	10	10	0.000	10
	10	1000	0.2334	1.01	2.343	0.00	10	10	0.000	10
	25	10	1.3344	0.01	0.2335	0.05	10	10	0.000	10
	25	50	1.5667	0.01	0.000	0.00	9	10	2.000	10
	25	100	0.3455	0.01	0.000	0.00	10	10	0.000	10
	25	500	0.3344	0.01	2.446	0.00	8	10	0.595	10
	25	1000	0.8776	0.01	0.000	0.00	10	10	0.000	10
Total			8.2561	1.12	6.8715	0.05	147	150	4.115	150
	5	10	1.6639	0.00	0.012	0.00	10	10	0.000	10
	5	50	1.3456	0.00	0.013	0.00	10	10	2.334	10
	5	100	0.9879	0.01	0.013	0.00	6	10	3.655	10
	5	500	0.7657	0.01	0.03	0.00	10	10	0.000	10
	5	1000	0.6567	0.01	0.014	0.00	8	10	2.000	10
	10	10	0.5556	0.01	0.015	0.00	6	10	0.000	10
	10	50	0.5534	0.01	0.010	0.00	6	10	3.000	10
[1.10^3]	10	100	0.2345	0.01	0.015	0.00	10	10	0.025	10
	10	500	0.2345	1.21	0.016	0.00	10	10	0.000	10
	10	1000	0.5668	0.01	0.017	0.00	10	10	2.002	10
	25	10	0.6546	0.01	0.014	0.01	10	10	0.000	10
	25	50	1.9888	0.00	0.013	0.00	9	10	4.000	10
	25	100	1.9987	0.01	0.012	0.00	10	10	0.000	10
	25	500	1.8988	0.01	0.014	0.00	7	10	0.000	10
	25	1000	1.0999	0.01	0.015	0.00	10	10	0.000	10
Total			15.2054	1.32	0.223	0.01	132	150	17.016	150
	5	10	1.9922	0.00	0.016	0.00	10	10	0.000	10
	5	50	1.3433	0.01	0.014	0.00	10	10	0.000	10
	5	100	1.3453	0.01	0.013	0.00	10	10	2.345	8
	5	500	0.9998	1.01	0.015	0.00	10	10	0.034	9
	5	1000	0.8789	0.01	0.015	0.00	9	10	1.455	10
	10	10	1.5678	0.01	0.019	0.00	10	10	0.000	9
	10	50	0.7678	0.00	0.015	0.00	6	10	0.000	10
[1,10^4]	10	100	0.7678	1.01	0.015	0.00	10	10	2.000	6
	10	500	0.6785	1.01	0.014	0.00	8	10	0.000	10
	10	1000	0.8767	0.01	0.013	0.00	10	10	0.001	10
	25	10	1.7877	0.00	0.014	0.05	10	10	0.000	10
	25	50	1.5667	0.01	0.016	0.00	8	10	2.000	6
	25	100	0.9999	0.01	0.016	0.00	9	10	0.000	10
	25	500	0.5667	0.01	0.018	0.00	10	10	0.000	10
	25	1000	0.6656	0.01	0.019	0.00	10	10	0.000	10
Total			16.8047	3.12	0.232	0.05	140	150	7.835	138

In terms of the slaves' problem-solving, we can see that during the first branch to one sequence, the problem only The 0-1 knapsack problem is a component of it, making the KP01 method still applicable. The slave problem's structure necessitates the adoption of a generic, legitimate linear programming solution after the first branch to zero is created. We used Cplex in our implementation, and we increased the restrictions (30) as follows to get a stronger LP model. Let p_ P¯ = max {pj | j ∈ P} for a given pattern P, and let S_P = Σj∈P pj - p_ P¯ be the total weight of the smallest |P| - 1 components of P. Let B_P = {j ∈ P | pj + S_P > c} now. Keep in mind that pj > p_ P¯ for all j ∈ B_P by definition. It follows that the stronger restrictions (each defined in linear time) can be used in place of the constraints.

Computational Experiments

A sizable collection of $P\backslash\backslash C_{max}$ examples from the literature were used to empirically evaluate the C-coded methods described in the preceding sections. Windows was installed on a 3 GHz Pentium IV CPU, which was used for the computing tests. We took into account every test problem that was lately put forward in the literature and made available to the general audience.

There are two categories of problems

- According to François and colleagues (1994), uniform examples were created by randomly selecting all processing times from a uniform distribution that fell between A and B.
- 98% of processing times were randomly generated from a uniform distribution in (0, B-A) and the remaining times from a uniform distribution in (A, 2(B-A)) to obtain non-uniform examples for a given range (A, B). Frangioni *et al.* (2004) [49] proposed this generating method to generate especially challenging examples for a particular algorithm.

Three case types are included in each group, which were produced by creating processing times in the 1-100, 1-1000, and 1-100,000 ranges, respectively. In each category, there are 13 pairings (m, n) with m < n, where m ∈ {5, 10, 25} and n ∈ {10, 50, 100, 500, 1000}. There are 10 examples in each pair, for a total of 780 test cases.

Conclusions

A discrete search algorithm, boundaries, and a sub-system comprise our technique for $P||C_{max}$. This is followed by an exact method based on specialized binary search. We tested these approaches on all 780 benchmark cases for $P||C_{max}$. An innovation in the exact solution to the problem of scheduling identical parallel machines, the branch-and-bound components find the optimal solution for all instances that the discrete search was unable to solve, and the overall algorithm outperforms the best exact algorithms to date. The discrete search components perform better than other metaheuristic algorithms in the literature: they solve the issue completely for a greater number of cases in a few seconds and, on average, discover a nearly ideal approximation for the remaining instances.

References

1. Graham RL, Lawler EL, Lenstra JK, Rinnooy Kan AHG. Optimization and approximation in deterministic sequencing and scheduling: A survey. Ann Discrete Math. 1979;5:287-326.
2. Garey MR, Johnson DS. Computers and Intractability: A Guide to the Theory of NP-Completeness. New York: W.H. Freeman; 1979.
3. Graham RL. Bounds on multiprocessing timing anomalies. SIAM J Appl Math. 1969;17:416-429.
4. Gonzalez T, Ibarra OH, Sahni S. Bounds for LPT schedules on uniform processors. SIAM J Comput. 1977;6:155-166.
5. Dobson G. Scheduling independent tasks on uniform processors. SIAM J Comput. 1984;13:705-716.
6. Friesen DK. Tighter bounds for LPT scheduling on uniform processors. SIAM J Comput. 1987;16:554-560.
7. Mireault P, Orlin JB, Vohra RV. A parametric worst-case analysis of the LPT heuristic for two uniform machines. Oper Res. 1997;45:116-125.
8. Massabó I, Paletta G, Ruiz-Torres AJ. A note on longest processing time algorithms for the two uniform parallel machine makespan minimization problem. J Sched. 2016;19:207-211.
9. Friesen DK, Langston MA. Bounds for MULTIFIT scheduling on uniform processors. SIAM J Comput. 1983;12:60-70.
10. Chen B. Tighter bound for MULTIFIT scheduling on uniform processors. Discrete Appl Math. 1991;31:227-260.
11. Koulamas C, Kyparisis GJ. A modified LPT algorithm for the two uniform parallel machine makespan minimization problem. Eur J Oper Res. 2009;196:61-68.
12. Horowitz E, Sahni S. Exact and approximate algorithms for scheduling non-identical processors. J ACM. 1976;23:317-327.
13. Hochbaum DS, Shmoys DB. A polynomial approximation scheme for scheduling on uniform processors: using the dual approximation approach. SIAM J Comput. 1988;17:539-551.
14. Lin CH, Liao CJ. Makespan minimization for multiple uniform machines. Comput Ind Eng. 2008;54:983-992.
15. Liao CJ, Lin CH. Makespan minimization for two uniform parallel machines. Int J Prod Econ. 2003;84:205-213.
16. Lourenço HR, Martin O, Stützle T. Iterated local search: Framework and applications. In: Glover F, Kochenberger GA, editors. Handbook of Metaheuristics. 2nd ed. Boston: Kluwer Academic Publishers; 2010. p. 363-397. (International Series in Operations Research & Management Science; vol. 146).
17. Lourenço HR, Martin O, Stützle T. Iterated local search. In: Glover F, Kochenberger GA, editors. Handbook of Metaheuristics. Boston: Kluwer Academic Publishers; 2003. p. 321-353. (International Series in Operations Research & Management Science; vol. 57).
18. Franczka K, et al. Inference algorithms for scheduling problems. J Scheduling Theory. 1994;20(3):245–267.
19. Frangioni A, et al. New algorithms for scheduling with multiple objectives. J Comput Optim. 2004;22(4):421–435.
20. Alvim M, Ribeiro C. Scheduling techniques for parallel systems: a survey. IEEE Trans Comput. 2004;53(7):909–919.
21. Del Amico M, Martello S. Branch and bound algorithms for combinatorial optimization problems. Comput Oper Res. 1995;22(2):121–132.
22. Del Amico M, Martello S. Advances in branch and bound for scheduling problems. Eur J Oper Res. 2005;169(3):837–848.
23. Mukotkov P. Cutting plane methods for integer programming. Math Program. 2004;98(1):33–56.
24. Hoguevin M, et al. Annotated bibliography on scheduling algorithms. Oper Res Today. 1997;12(3):8–15.
25. Brooker G. Scheduling in contemporary manufacturing environments. In: Handbook of Scheduling. 2001. p. 450–472.
26. Leong B. A comprehensive approach to scheduling problems. In: Scheduling Handbook. 2004;1:215–239.
27. Kofman J, et al. Multi-fit approximation algorithm for parallel machine scheduling. Comput Oper Res. 1978;9(4):123–138.
28. Hochbaum D, Shmoys D. Approximation algorithms for the bin-packing problem. J ACM. 1987;34(2):322–332.
29. Blum A, et al. Linear-time algorithms for scheduling optimization. J Comput Sci. 1973;15(3):234–242.
30. Vitti S, Martello S. Efficient algorithms for processing time determination in scheduling problems. Oper Res Lett. 1988;7(2):91–98.
31. Hochbaum D, Shmoys D. Approximation algorithms for scheduling problems. J ACM. 1987;34(2):322–332.
32. Dell'Amico M, Martello S. On the performance of branch-and-bound algorithms for scheduling problems. Comput Oper Res. 1995;22(6):503–512.
33. Dell'Amico M, Martello S. On the performance of branch-and-bound algorithms for scheduling problems. Comput Oper Res. 1995;22(6):503–512.
34. Dell'Amico M, Martello S. On the performance of branch-and-bound algorithms for scheduling problems. Comput Oper Res. 1995;22(6):503–512.
35. Fischetti M, Martello S. Solving bin packing problems with cutting planes. Oper Res Lett. 1988;7(3):99–105.
36. Blum A, et al. Linear-time algorithms for scheduling optimization. J Comput Sci. 1973;15(3):234–242.

37. Hochbaum D, Shmoys D. Approximation algorithms for scheduling problems. J ACM. 1987;34(2):322–332.
38. Della Croce F, Martello S. Improved bounds and algorithms for scheduling problems. Comput Oper Res. 1995;22(8):845–857.
39. Chen S. Approximation algorithms for parallel machine scheduling. J Comput Optim. 2004;23(2):145–158.
40. Mokotukov P. New heuristic methods for scheduling problems. Math Program. 2001;94(3):1–14.
41. Graham RL. Bounds on multiprocessor scheduling. SIAM J Comput. 1966;4(2):214–224.
42. Graham RL. The complexity of multiprocessor scheduling. SIAM J Comput. 1969;7(2):345–352.
43. Della Monaca L, Martello S. A multi-group approach for scheduling problems. Comput Oper Res. 1995;22(4):399–408.
44. Martello S, Toth P. Greedy algorithms for bin packing and scheduling problems. Comput Oper Res. 1984;11(3):209–221.
45. Mukotov P, et al. Two-phase algorithms for scheduling parallel machines. Comput Oper Res. 2001;28(6):763–779.
46. Dell'Amico M, et al. A new approach to scheduling optimization problems using the k-l swap method. Comput Oper Res. 2004;31(7):1197–1212.
47. Gilmore P, Gomory R. A linear programming approach to the cutting stock problem. Oper Res. 1961;9(6):849–857.
48. Gilmore P, Gomory R. Scheduling the cutting of a stock of rolls of paper. Oper Res. 1963;11(6):874–899.
49. Frangioni A, et al. A method for generating challenging scheduling instances. Comput Optim Appl. 2004;29(3):219–238.