A Hybrid Searching Method for the Unrelated Parallel Machine Scheduling Problem

Christoforos Charalambous[1], Krzysztof Fleszar[2], and Khalil S Hindi[2]

[1] Department of Computer Science and Engineering, Frederick University, Cyprus,
c.charalambous@frederick.ac.cy
[2] Olayan School of Business, American University of Beirut (AUB), Lebanon
kf09@aub.edu.lb, khalil.hindi@aub.edu.lb

Abstract. The work addresses the NP-hard problem of scheduling a set of jobs to unrelated parallel machines with the overall objective of minimizing makespan. The solution presented proposes a greedy constructive algorithm followed by an application of a Variable Neighborhood Decent strategy that continually improves the incumbent solution until a local optimum is reached. The strength of the approach lies in the adoption of different objectives at various stages of the search to avoid early local optimum entrapment and, mainly, in the hybridization of heuristic methods and mathematical programming for the definition and exploration of neighborhood structures. Experimental results on a large set of benchmark problems attest to the efficacy of the proposed approach.

Key words: parallel machine scheduling, hybrid optimization, variable neighborhood search, mixed-integer programming

1 Introduction

The unrelated parallel machines scheduling problem is that of scheduling without preemption n jobs available at time zero on m unrelated machines to minimize the makespan (maximal machine completion time), $C_{\max}$. Henceforth, we will also use the term 'span' to signify the completion time of a machine.

The machines are unrelated in that the processing time of a job depends on the machine to which it is assigned. The practical importance of the problem stems from the fact that it is common in many industrial applications to have parallel resources with different capabilities, perhaps procured at different times. These resources would be capable of carrying out the same tasks, but the time taken to perform a task would depend on the resource on which it is performed. Such applications can be found in industries like painting, plastic, textile, glass, semiconductor, chemical, and paper manufacturing, as well as in some service industries [1].

Literature on parallel machine scheduling problems is extensive; for general references and surveys, see [2, 3, 5, 7, 10]. However, most of the literature addresses identical machines, where the processing time of a job is the same regardless of the machine to which it is assigned, or uniform machines, where

the processing time of a job is proportional to the speed of the machine. Of the relatively small number of works that address unrelated machines, most deal with the case without setup times. For minimization of the makespan, Martello et al offered exact and approximation algorithms [6]; Mokotoff and Chretienne a cutting plane algorithm [8]; Mokotoff and Jimeno [9] heuristics based on partial enumeration; Ghirardi and Potts [4] a recovering beam search method; and Shchepin and Vakhania [11, 12] approximation algorithms.

Throughout, the following notation is used:

$N = \{1, \ldots, n\}$	set of jobs, n being the total number of jobs.
$M = \{1, \ldots, m\}$	set of machines, m being the total number of machines.
p_{jk}	processing time of job j on machine k
$M(j)$	the machine to which job j is assigned to.
J_k	set of jobs assigned to machine k.
C_k	span (completion time) of machine k.
$C_{\max} = \max_{k \in M} C_k$	makespan (maximum machine completion time).

2 Definition of objective criteria

The problem's main objective is to minimize the solution makespan (C_{max}). However, focusing solely on the main objective may inhibit the development of the search process as other desirable characteristics are likely to be ignored. In this work, in addition to the main objective, auxiliary criteria are employed to assist the development of high quality solutions. The criteria used are:

1. **Sum of spans:** To be able to minimize the makespan it is helpful to associate jobs to machines that process them efficiently. The lower the sum of spans (an average measurement on how efficiently the jobs have been assigned), the more slack can be expected on non-makespan inducing machines, leading to an increased capacity for solution improvement through job reassignments.
2. **Number of makespan-inducing machines** Neighborhood search structures operate by altering incumbent solutions through a small number of job reassignments aiming at an overall improvement. The fewer the number of machines that have a span equal to the makespan, the more probable the existence of a job reallocation pattern that would reduce the makespan.

The mode in which objective criteria are used depends on the neighborhood structure, as described in Section 4.

3 Initial solution

Numerous constructive algorithms have been proposed in the literature including the efficient APPROX method of Martello [6]. These algorithms focus on finding techniques for developing solutions that minimize the objective function. In this work the primal goal of the solution construction is not to minimize makespan but rather to seed the searching phase of the algorithm with a solution that resides in a promising region. The rational against opting for makespan

minimization is that fast constructive algorithms invariably adopt a greedy approach in the solution construction that often opts for short-sighted moves which worsen the objective function least. Such an approach leads to relatively good solutions but, due to the naive nature of choices, there is little room left for further improvements.

The initial solution is calculated based on the linear relaxation of the UPMSP

$$\min C_{max} \tag{1}$$

subject to

$$\sum_{k \in M} x_{jk} = 1 \;\; \forall j \in N \tag{2}$$

$$C_{max} \geq \sum_{j \in N} p_{jk} x_{jk} \;\; \forall k \in M \tag{3}$$

$$x_{jk} \in \{0, 1\} \;\; \forall j \in N, k \in M \tag{4}$$

where x_{jk} is a binary variable signifying whether job j is assigned to machine k.

When the integrality constraint in 4 is relaxed, the solution generated by the solver will consist of a large set of x_{jk} variables set to zero, and the rest being either one or set to a fractional value. The initialization heuristic sets the upper bound of all x variables that have value zero to zero (i.e. forbidding the corresponding assignment). Subsequently, all fractional variables are sorted based on *regret*. The regret of a variable is defined as the added processing time that would incur in the solution if the corresponding variable was set to one and is calculated by $p_{jk}(1-x_{jk})$. The variable with the maximum regret has its upper bound set to zero and the model is resolved until no variable has a fractional value. Note that if the model has no fractional variables, the corresponding solution is feasible.

4 Neighborhood Moves and Variable Neighborhood Decent

Improving an incumbent solution can often be achieved through the reassignment of jobs to machines. The means of these reassignments, henceforth called moves, is central to the definition of the neighborhood structures used in the searching approach. All moves are based on a series of additions (allocating a job to a machine) and removals (deallocating a job from a machine) and can be categorized as follows:

Transfer removes job j from its machine, $M(j)$, and adds it to machine $k \neq M(j)$ (one removal and one addition).

Swap removes jobs i and j ($i \neq j$) from their respective machines ($M(i) \neq M(j)$), and adds i to $M(j)$ and j to $M(i)$ (two removals and two additions).

Closed ejection chain involves a closed chain of $h \geq 3$ machines. A job is removed from each machine of the chain and added to the next machine in the chain (h removals and h additions).

Open ejection chain involves an open chain of $h \geq 3$ machines. A job is removed from each but the last machine of the chain and added to the next machine in the chain ($h-1$ removals and $h-1$ additions).

Compound move is any collection of transfers, swaps, closed, and open ejection chains such that on each machine at most one addition (restriction 1) and at most one removal (restriction 2) is performed (h removals and h additions, $1 \leq h \leq m$). Furthermore, an expanded compound move can be effected if restriction 1 or restriction 2 is removed, but not both.

The valid moves of a neighborhood define its structure and size. The simpler the moves that can be employed the faster the exploration of the neighborhood. On the other hand, simpler structures are less likely to be capable of identifying improving moves than more involved ones.

The proposed algorithm defines three neighborhood structures of increasing complexity named small (SNS), medium (MNS) and large (LNS). SNS is applied first and upon local optimum entrapment the search progresses to the MNS, altering also the objective criterion. When MNS also reaches a local optimum, LNS is employed. If one iteration of LNS leads to an improvement, the cycle is repeated otherwise the incumbent solution is the algorithm's proposed solution.

4.1 Small Neighbourhood Search

The initial solution is often quite poor but with high potential, given the high slacks. Improvements are obtained through a small neighborhood structure that is searched exhaustively, adopting the best improving move each time. In SNS, the neighborhood structure is defined as any transfer or swap that involves the machine inducing the makespan (in case more that one machines have spans equal to the makespan, one is randomly designated as the makespan machine).

A move is considered improving if, for all the machines affected by it, the new span is smaller than the incumbent makespan. Given the set of improving moves of a small neighborhood, the best is considered the one that minimizes the sum of spans (see Section 2). To further assist the guidance of the search to promising regions, in cases where no improving move can be found through transfers or swaps, or where the move increases the total span to more than 10% of the lower bound, the neighborhood structure is augmented to include ejection chains (open or close) of size 3. SNS terminates when no improving move can be found.

The adoption of a dynamic neighborhood structure proved necessary to balance the effectiveness and efficiency of the search. Exhaustively examining transfer and swap moves can be done very fast, given the simplicity of the moves. If a satisfactory move is identified, something likely in the early stages, it is adopted, thus allowing a speedy convergence without sacrificing the solution characteristics. Only upon non-satisfactory moves is the, more involved, ejection chain move examined.

4.2 Medium Neighbourhood Search

Focusing only on moves that involve the makespan machine inhibits the search from identifying more promising solutions. Clearly, moves in which the makespan machine is not involved cannot improve the overall objective of makespan minimization. However, moves involving the remaining machines can lead to improvements by increasing slack space (reducing the sum of spans) or reducing the number of makespan machines, both of which are desirable as explained in Section 2. In MNS, valid moves include any transfer or swap between any pair of machines. A move is considered undesirable if it worsens any of the two auxiliary objectives. MNS adopts a first-improving approach, committing the first move identified that reduces the sum of spans without increasing the number of makespan machines. Using a first-improving instead of a best-improving approach and limiting the moves to transfers and swaps was preferred to economize on computational effort, given the increase in the set of machines that could be involved in MNS moves.

4.3 Large Neighbourhood Search

As explained, neighborhood structures that employ limited scope moves have limited capabilities of yielding improvements. This is particularly true when the incumbent solution is relatively tight, as solutions obtained after SNS and MNS are likely to be. To escape from such solutions, a large neighborhood structure is defined that explores all feasible compound moves.

Such structures can identify complicated moves that may yield improvements but the difficulty lies in the development of efficient mechanisms for facilitating the search. To achieve this a MILP model is developed as described below.

$$\min C_{max} + \sum_{k \in M} \frac{C_k}{1000|M|} \tag{5}$$

subject to

$$\sum_{j \in N} y_{jk} \leq 1 \quad \forall k \in M \tag{6}$$

$$\sum_{j \in J_k} \sum_{l \in M} y_{jl} \leq 1 \quad \forall k \in M \tag{7}$$

$$C_k = C'_k + \sum_{j \in J} p_{jk} y_{jk} - \sum_{j \in J_k} \sum_{l \in M} p_{jk} y_{jl} \quad \forall k \in M \tag{8}$$

$$C_{max} \geq C_k \quad \forall k \in M \tag{9}$$

$$C_{max} \leq C'_{max} - 1 \tag{10}$$

where
y_{jk} — boolean variable set to 1 if job j is moved to machine k
C'_k — incumbent span of machine k
C'_{max} — incumbent makespan

The objective function (equation 5) minimizes the makespan using also the auxiliary objective of sum of spans minimization as a secondary objective. Constraint 6 ensures that at most one job can be added to any machine whereas constraint 7 limits the number of jobs that can be removed from a machine to one. Constraint 8 calculates the new span of each machine by adding to the incumbent span the processing times of all jobs moved to that machine and subtracting the processing times of all jobs that were allocated to the machine and have been moved to another machine. Constraint 9 links the makespan to the new spans and constraint 10 ensures that the makespan is improved.

LNS commences by restricting the compound moves to those that allow at most one insertion and one removal on each machine. If no such move is found, the constraint 6 is relaxed, allowing for more than one job to be added to a machine. If again no improving compound move is identified, constraint 6 is reinstated and constraint 7 is relaxed, allowing for more than one jobs to be removed from a machine but limiting the jobs to be added to a machine to one.

Given the large number of binary variables defined by the model, it is crucial to reduce as many as possible so that the model can be solved by a MP solver in realistic execution times. The following reductions are defined:

- $y_{jk} = 0 \quad \forall j \in J_k, k \in M$
- Assuming p_k^{max} represents the largest processing time on machine k of the jobs in J_k, then $y_{jk} = 0 \ \forall j \in N$ if $C'_k + p_{jk} - p_k^{max} > C'_{max} - 1$
- Given two jobs $i, j \in J_k$, $y_{j,l} = 0 \ \forall l \in M \backslash \{k\}$ if $p_{ik} \geq p_{jk} \wedge p_{il} \leq p_{jl}$

The first reduction ensures that a job cannot be moved to its current allocation. The second reduction reduces a move if even assuming the removal of the job with the largest processing time from the destination machine, the move would still be bigger than or equal to the incumbent makespan. Finally, the third reduction states that if two jobs are currently assigned to the same machine, then if the first has bigger or equal processing time on the incumbent machine and for a given destination machine it has less or equal processing time, such a move would always be preferable therefore the variable corresponding to moving the second is reduced. Note that if constraint 7 is relaxed, the second and third reductions do not apply.

Since the incumbent solution is likely to be tight given the prior application of SNS and MNS, the amount of variables reduced is high and experimentation has shown that relevant models can be addressed within acceptable execution times. As a safety net, the time assigned to the MP solver for a model instance has been capped to 1 second.

5 Computational results

The proposed algorithm was implemented in C#.NET. The MIP problems were solved using ILOG CPLEX version 12.1, with the number of threads limited to 1 and the relative and absolute gaps set to zero. All tests were executed on a x5570 2.93GHz Intel Xeon processor using a single thread on a single core.

Three classes of random problem instances were generated, adopting the same pattern as that used in the literature. Specifically, for the first two classes processing times were taken randomly from uniform distributions $U(10, 100)$ and $U(10, 1000)$, respectively. For the third, *speeds* are assigned to machines and *lengths* to jobs and a weighted randomized processing time is calculated for each job-machine pair (see. [4] for more details).

For each problem class, 20 different instances were generated with the machine number obtained from $\{10, 20, 30, 40, 50\}$ and the job number from $\{100, 200, 500, 1000\}$. Computational results aggregated over classes, machines, and jobs are provided in Table 1. All presented results refer to the average over the 20 instances. The proposed algorithm was compared against the APPROX algorithm of Martello et al [6] and the Recovering Beam Search method with beam width 3 of Ghirardi and Potts [4] (denoted by RBS). For each algorithm the deviation from the lower bound (obtained through applying Langrangean relaxation on the UPMSP model) and the average and maximum execution time are provided. Finally, the last table column (denoted as 'Imp') shows the percentage of improvement (or deterioration) of the proposed solution against the best known deviation. For APPROX and RBS the deviation and times presented in [4] were used (implemented in C++ and executed on a PC-Pentium III/866).

Table 1. Comparative results on set of problem instances

Aggr. by		APPROX			RBS			VND			
		Dev(%)	avT(s)	maxT(s)	Dev(%)	avT(s)	maxT(s)	Dev(%)	avT(s)	maxT(s)	Imp(%)
cl.	1	6.1	17.7	83.0	5.1	79.6	392.2	4.0	1.3	3.8	31.7
	2	12.0	13.5	61.9	10.7	74.0	351.5	10.1	1.7	5.4	25.2
	3	3.1	54.6	212.4	2.1	92.7	427.8	1.4	2.5	7.8	51.2
m	10	1.9	22.8	174.4	1.2	63.8	262.1	0.9	0.8	2.1	41.5
	20	4.0	24.4	176.1	3.0	65.3	260.4	2.3	1.4	3.2	40.5
	30	6.5	28.0	181.8	5.2	75.1	291.2	4.7	2.0	5.2	33.6
	40	9.7	33.0	201.7	8.3	98.9	391.7	7.0	2.3	6.6	35.5
	50	13.2	34.8	212.4	12.1	107.4	427.8	11.0	2.6	7.8	29.0
n	100	17.2	< 1	< 1	15.7	< 1	< 1	15.3	0.4	1.1	7.8
	200	6.6	1.4	3.1	5.2	1.8	3.7	4.6	1.0	2.4	22.2
	500	2.7	15.5	31.3	1.9	29.3	41.9	1.0	2.1	4.1	52.8
	1000	1.7	97.0	212.4	1.1	285.7	392.2	0.4	3.8	7.8	61.2
overall		**7.1**	**28.6**	**212.4**	**6.0**	**82.1**	**427.8**	**5.2**	**1.8**	**7.8**	**36.0**

As shown, the proposed VND greatly outperforms the other algorithms when compared by either class, machines or jobs. VND performs best in the third class which also has the highest time demands. Comparing by number of machines, it seems that VND consistently outperforms the other algorithms independent of machine size. On the contrary, when comparing by job number, it is evident that the performance improvement is linked with the problem size. Also, taking into account processor performance benchmarks, VND achieves comparable execution times with RBS.

6 Conclusion

A variable-neighborhood descent, hybridized with mathematical programming elements, that addresses the NP-hard problem of scheduling unrelated parallel machines with the objective of minimizing the makespan, has been presented. Neighborhood search structures, of increasing complexity, are investigated to guide the search to promising regions. Local optimum entrapment is avoided through the use of large neighborhoods which are explored with the use of mathematical programming. The performance of the proposed scheme has been evaluated by comparing its solutions on a set of benchmark problem instances to those generated by the best available algorithms. The results show that the proposed scheme provides considerably better solutions.

References

1. JP Arnaout, G. Rabadi, and Rami Musa. A two-stage ant colony optimisation algorithm to monomize the makespan of unrelated parallel machines with sequence-dependent setup times. *Journal of Intelligent Manufacturing*, 2009. forthcoming.
2. J. Blazewicz, K. H. Ecker, E. Pesch, G. Schmidt, and J. Weglarz. *Scheduling computer and manufacturing processes*. Springer, Berlin, 1996.
3. P. Brucker. *Scheduling algorithms*. Springer, Berlin, 2004.
4. M. Ghirardi and C. N. Potts. Makespan minimization for scheduling unrelated parallel machines: A recovering beam search approach. *European Journal of Operational Research*, 165(2):457–467, 2005.
5. C-Y. Lee and M. Pinedo. Optimization and heuristics of scheduling. In P. M. Pardalos and M. G. C. Resende, editors, *Handbook of applied optimization*. Oxford University Press, New York, 2002.
6. S. Martello, F. Soumis, and P. Toth. Exact and approximation algorithms for makespan minimization on unrelated parallel machines. *Discrete Applied Mathematics*, 75:169–188, 1997.
7. E. Mokotoff. Parallel machine scheduling problems: A survey. *Asia-Pacific Journal of Operational Research*, 18(2):193–242, 2001.
8. E. Mokotoff and P. Chretienne. A cutting plane algorithm for the unrelated parallel machine scheduling problem. *European Journal of Operational Research*, 141(3):515–525, 2002.
9. E. Mokotoff and J. L. Jimeno. Heuristics based on partial enumeration for the unrelated parallel processor scheduling problem. *Annals of Operations Research*, 117(1–4):133–150, 2002.
10. M. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Prentice Hall, New Jersey, 2 edition, 2002.
11. E. Shchepin and N. Vakhania. An absolute approximation algorithm for scheduling unrelated machines. *Naval Research Logistics*, 53(6):502–507, 2006.
12. E. V. Shchepin and N. Vakhania. An optimal rounding gives a better approximation for scheduling unrelated machines. *Operations Research Letters*, 33(2):127–133, 2005.