HAL
open science

VNS/TS for a parallel machine scheduling problem

Marc Sevaux, Kenneth Sörensen

To cite this version:

Marc Sevaux, Kenneth Sörensen. VNS/TS for a parallel machine scheduling problem. Proceedings of the 18th Mini EURO Conference on Variable Neighborhood Search (VNS), 2005, France. pp.1-10. hal-00019096

HAL Id: hal-00019096

https://hal.science/hal-00019096v1

Submitted on 15 Feb 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

VNS/TS for a parallel machine scheduling problem

Marc Sevaux[1], Kenneth Sörensen[2]

[1]University of South-Brittany
LESTER – CNRS, FRE 2734 – Centre de Recherche – BP 92116 – F-56321 Lorient – France
marc.sevaux@univ-ubs.fr

[2]University of Antwerp
Faculty of Applied Economics – Prinsstraat 13 – B-2000 Antwerpen, Belgium
kenneth.sorensen@ua.ac.be

Abstract

The goal of this work is to develop an efficient metaheuristic to solve the parallel machine scheduling problem where the objective function is to minimize the weighted number of late jobs. This problem appears in several real-life applications and particularly in finance and production management. The weighted number of late jobs can also be seen as an indicator of the quality of a solution or as a minor objective in multi-objective optimisation. To our knowledge, only a limited number of methods have been developed to solve this problem. The best-performing method to date is based on constraint propagation techniques and does not succeed in optimally solving instances of fifty jobs with more than sixty percent success rate. One of the aims of this research is to provide an efficient method to solve problems with up to 100 jobs at least.

A problem-specific representation is used to represent solutions. It consists of a priority list of late jobs and a list of early and sequenced jobs for each machine. This representation is well-suited for this objective and easy to manipulate with any computer language.

Based on this representation, a set of neighbourhoods is used in a VNS strategy. To obtain good results a tabu search procedure is embedded as the local search mechanism. Numerical experiments are conducted on a set of instances of 50 jobs and compared with an exact procedure. Results prove that the proposed method can indeed be used to find solutions of large instances.

Keywords: VNS, Tabu search, parallel machines, late jobs.

1 Introduction

In a scheduling environment, a set of m parallel and identical machines is available for processing jobs. Each of a set of n jobs has to be sequenced on one of the m machines. The *processing time* of job j is denoted by p_j and its *weight* by w_j. Each job j should be processed within its time window $[r_j, d_j]$ (where r_j is the *release date* and d_j is the *due date*). Of course, the start of a job cannot be scheduled before its release date. If a job cannot be finished before its due date, it is considered *late*.

The objective is to minimise the weighted number of late jobs ($\sum w_j U_j$, where U_j is a binary value equal to 1 if job j is late and 0 otherwise). This problem, denoted $Pm|r_j|\sum w_j U_j$ in the standard classification [6], is $\mathcal{NP}$-hard in a strong sense [5].

An important remark is the following:

Remark 1 *Late jobs can be scheduled arbitraryly after early jobs in any optimal solution. From a practical point of view, late jobs do not need to be scheduled.*

In [1], a complete exact method is developed for solving the $Pm|r_j|\sum w_j U_j$ problem. This method is based on a constraint propagation technique which uses new lower bound schemes and new algorithms for propagation. This method will further be denoted by CPP. Constraint propagation algorithms solve the decision version of combinatorial optimization problems and dichotomy is used to find the optimal solution. At each step of this approach, an upper bound UB and a lower bound LB are computed. A decision variant of the problem is then solved using the mean value of UB and LB (MV = (UB − LB)/2) as a reference point. If a feasible schedule exists whose objective value is MV, then MW becomes the new UB and the process is started again. If there is no schedule with objective value MV, then MV becomes the new lower bound and the process is started again. However, this type of approach is unable to solve large instances efficiently. Even with a time limit of 10 minutes, instances with more than 50 jobs cannot be solved to optimality. Moreover, when a dichotomy technique fails to find a solution within the time limit, it often does not have enough time to solve one decision version of the problem, and will return a very poor solution.

A Lagrangian relaxation algorithm [3] is able to solve the same problem but again with not more than 50 jobs. Since better results are provided by the CPP approach, this method will not be considered in the numerical experiment section. We refer the reader to [3] for a comparison of the two methods presented above.

2 A VNS-TS algorithm for parallel machine scheduling

The proposed approach consists of a tabu search (TS) technique imbedded as a local search operator within a variable neighbourhood search (VNS) method. The encoding of a solution is specific to the problem and its irregular objective (see e.g. [4]). The property mentioned in remark 1 is used for this particular encoding. A set of nested neighbourhoods is developed and used for both the TS and the VNS. The TS always performs a complete exploration of the current neighbourhood whereas the VNS accepts the first improving solution.

2.1 Encoding

Remark 1 states that late jobs do not need to be scheduled. Hence, an easy way to represent a solution is using m lists of jobs that are effectively sequenced on the different m machines with their starting times. The starting times for each early job are determined by the main algorithm and will be explained later in the paper. Maintaining a feasible solution for each machine individually ensures that the jobs on each machine can be processed without violating the classical scheduling constraints.

To complete the solution and avoid non-necessary computations, a list of not-yet scheduled jobs is maintained. This list corresponds to the list of late jobs. According to remark 1, no starting times are given for these jobs.

2.2 Neighbourhood structures

Since the objective of our problem is to minimize the number of late jobs, the quality of a solution can only be improved by moving jobs (named candidate jobs) from the *late* list to one of the m *machine* list. To do so and according to the encoding presented above, a starting time should be found for the candidate jobs. If there is enough free processing time available on one of the machines (e.g. idle times due to the release dates), a candidate job can be moved from the late list to one of the machine lists without violating any constraints. If not, a late job can only be moved to a machine by removing one or several consecutive early jobs from this machine and putting the job in their place on the considered machine.

In our algorithm, only one late job is exchanged at a time. The reason for this is that exchanging

several late jobs at the same time increases the complexity of the algorithm, while making it more difficult to determine the neighbourhood. With this exchange rule, we define $\mathcal{N}_k$, our *neighbourhood of size* k as *the set of feasible solutions that result from swapping one candidate job with 0 to up to* k *consecutive early jobs on the same machine.*

Figure 1 presents an example with 2 machines and 9 jobs. In the solution represented in this figure, jobs 2 and 3 are late, jobs 1, 6, 5 and 9 are sequenced in this order on machine M1 and jobs 8, 4 and 7 are sequenced in this order on machine M2. Of course, jobs on the machines are early. If we assume that we are determining the neighbourhood with $k = 2$, we take the first candidate (late) job (job 2) and try to set it early on machine M1 and M2 in any possible position, without removing any jobs from these machines. If it can be set early, without removing any job, we have found one or more feasible solutions in the neighbourhood (any position in which the job can be inserted constitutes a new solution in the neighbourhood). Then, we try to set this candidate job early by setting late one of the early jobs. So we try to set job 2 early by setting late succcessively jobs 1, 6, 5 and 9 on machine M1, possibly generating new solutions in the neighbourhood. We redo this with machine M2. Finally, we try to set this candidate job early by setting late two (k) consecutives early jobs. So we try to set early candidate job 2 by setting late jobs 1 and 6, then by setting late jobs 6 and 5,then with jobs 5 and 9 on machine M1. And the same is done with jobs 8, 4 and with jobs 4 and 7. The complete neighbourhood is found by repeating this process for late job 3.

Late jobs	{2, 3}

M1	{1, 6, 5, 9}
M2	{8, 4, 7}

Figure 1: Example with 2 machines and 9 jobs

All these "moves" have a specific cost (variation of the objective function value). This cost is stored with the neighbour solution and will be used in VNS and TS algorithm. Note that we never attempt to compact the schedules on the individual machines e.g. by moving all jobs forward as much as possible. This is left for future research.

Another option is to exchange one late job with exactly k early jobs ($k \in [0, n]$). During a preliminary computational experiment phase, this option has been tested and shown to be inefficient.

2.3 Initial solutions

We use a greedy approach to construct an initial solution. Initially, all jobs are considered late, and the list of late jobs is sorted in increasing order of the ratio w_j/p_j (weight over processing time). This sorting rule is denoted *weighted shortest processing time*. Jobs are then added to one of the machine lists in the order they appear on this list; machines are taken in sequencial order. This rule is efficient on single and parallel machine problems and was again used for our problem (see [8]).

In the computational test phase, we will also run the final algorithm from another inital solution. This initial solution will be obtained using the same greedy algorithm but a random order of the initial list of late jobs is used. As explain in section 3, several runs of the main algorithm will be performed and the best solution over all runs will be kept for comparison.

2.4 Tabu search

The tabu search is based on the same neighbourhoods and is briefly described here. A maximum of it_{max} iterations are performed during this phase. The tabu search works with neighbourhood $\mathcal{N}_k$ (k is given by the main VNS algorithm). The complete neighbourhood $\mathcal{N}_k$ is generated. The best solution from $\mathcal{N}_k$ is extracted and its tabu status is checked. If the solution is tabu and does not satisfy the aspiration criterion, the second best solution from $\mathcal{N}_k$ is used, etc, until all solutions from $\mathcal{N}_k$ are tested, otherwise, the solution is used as the new current solution and the tabu search continues.

The tabu status of a solution is defined by a pair of integer values: (1) the sum of the weights of the late jobs and (2) the sum of the processing times of the late jobs. This technique has been used to avoid similar configurations that can be found by *e.g.* just switching machines identifiers.

2.5 VNS

Our variable neighbourhood search algorithm follows the classical definition of the VNS algorithm proposed in [7]. The standard algorithm has been used (see algorithm 1), but a few shortcuts have been added such as loop exits (*e.g.* if a feasible solution can be found in which all jobs are early, which is of course optimal).

Algorithm 1: VNS main algorithm

```
while Stopping conditions are not met do
    k ← 0
    while k ≤ k_max do
        Shaking: Choose a random neighbour x' ∈ N_k(x)
        Local search: Apply tabu search on x' as defined previously and get x''
        Move or not: if x'' is better than x' then
            x ← x', k ← 0
        else
            k ← k + 1
```

3 Experiments

Instances for numerical experiments are taken from [1]. They have been generated for single machine problems in [2] but can be easily extended to more than one machine. Four important characteristics are taken into account for generation: (1) distribution of processing times, (2) distribution of weights, (3) overall load of the machines (load is computed as the ratio between the sum of processing times and $m(\max d_j - \min r_j)$ and (4) margin of each job $m_i = d_j - r_j - p_j$. Hence, processing times are generated by the uniform distribution $\mathcal{U}[p_{min}, p_{max}]$, weights are generated by $\mathcal{U}[1, w_{max}]$, release dates by the normal distribution $\mathcal{N}(0, \sigma)$ and margins by $\mathcal{U}[1, m_{max}]$. Since most release dates are in $[-2\sigma, 2\sigma]$, load can be approximated by $load = \frac{n(p_{min}+p_{max})/2}{m(4\sigma+p_{max}+m_{max})}$. The value of σ is computed using this formula. Parameters are taken from the table 1 and one instance is generated for each combination of these parameters.

Parameter	Possible values	Parameter	Possible values
n	50; 100	m_{max}	50; 200; 350; 500; 650
m	1; 3; 6	$load$	1.0; 1.6; 2.2
$[p_{min}, p_{max}]$	[0,100];[25,75]	w_{max}	1; 10; 100

Table 1: Parameters for instance generation

Two types of tests have been performed. First, we compare our resolution approach with the exact approach proposed in [1] on a number of 50-job instances. The number of optima found, as well as the improvements of one method over the other are reported. In the second type of tests, we use the 100-job instances and report the results of the main algorithm and compare it with the best of 10 runs on the same algorithm but starting from an inital random solution.

In Tables 2, 3 and 4, the first three columns ("Optima") present the number of optimal solutions found by the CPP method (see [2]), the number of optimal solutions found by the VNS method presented here and the average gap (in %) between the VNS solution and the optimal solution. The next two columns ("Best sol.") present results when the optimal solution is not known. Column "Improv." shows the number of times the best solution found by the CPP method is improved by our VNS and the improvement in percentage. The last two columns give the average CPU times of each method.

Inst. type	Optima			Best sol.		CPU times (s)	
w_{max}	# CPP	# VNS	Gap	Improv.	Gap	CPP	VNS
1	29	0	40.18	0	–	64.39	3.95
10	28	0	30.44	1	-65.23	101.58	4.54
99	28	0	33.84	0	–	176.91	4.65
Total	85	0	34.88	1	-65.23	114.29	4.38

Table 2: Results for $n = 50$ and $m = 1$

Clearly, for 50 jobs and one machine problems, the CPP method is adequate, 85 out of 90 optimal solution are found. Only one instance (for which optimal solution is not found) is improved by our approach; in that case, the CPP solution is improved by about 65%.

Inst. type	Optima			Best sol.		CPU times (s)	
w_{max}	# CPP	# VNS	Gap	Improv.	Gap	CPP	VNS
1	3	0	41.00	25	-62.56	549.88	2.01
10	4	0	36.08	23	-77.56	542.24	2.54
99	3	0	68.96	21	-77.63	554.13	2.50
Total	10	0	46.14	69	-72.15	548.75	2.35

Table 3: Results for $n = 50$ and $m = 3$

For three machine problems, the CPP has more difficulties to complete the exhaustive search, and only a small number of optimal solutions are found. As mentioned before, if the CPP search fails, in most cases, not even a single run of the decision problem is achieved and the quality of the solution is poor. This justifies the gaps observed in "Best sol." columns.

As the number of machines increases, the efficiency of our approach relative to the CPP method increases. In fact, the CPP method becomes more and more inefficient with increasing number of machines. This is probably due to a property of the CPP method, in particular the fact that this method is unable to detect similarities of configurations of early jobs. In our problem, all machines are identical but in the CPP method, setting job 1 and job 2 early on machine 1 is totally different

Inst. type	Optima			Best sol.		CPU times (s)	
w_{max}	# CPP	# VNS	Gap	Improv.	Gap	CPP	VNS
1	19	10	25.29	8	-69.25	255.21	0.86
10	15	11	0.00	9	-83.48	303.62	1.22
99	16	11	0.00	10	-71.46	283.65	1.17
Total	50	32	25.29	27	-74.81	280.82	1.08

Table 4: Results for $n = 50$ and $m = 6$

from setting these jobs early on machine 2. For our VNS/TS method, we can observe a computing time decrease as the number of machines increases. This is due to the fact that, the more machines we have, the shorter time the VNS/TS method needs to find an inproving solution and stop the search.

In Tables 5, 6 and 7, results are presented for instances with 100 jobs. There is no other competitive method that solve these instances. Hence, the VNS/TS is run from two different initial solutions. The first version uses an initial heuristic where jobs are sorted according to the weighted shortest processing time rule and then inserted in the machines in this order, the VNS/TS is applied from that solution. This method will be denoted BW (Best Weighted). The second version uses a random order rule for initial insertion and the VNS/TS is applied on that solution. This approach is run ten times and the best result is reported. We will denote this method RS (Random Shuffle). The first two columns give the number of times BW gives the best results as well as its improvement over RS. The next two columns give the same results for RS. The next three columns give specific results on BW: the number of times VNS/TS improves the initial solution; when improved, how much; and how long it takes. The last three columns report the same results for RS (CPU times are the average running times over all ten runs).

Inst. type	First place				BW + VNS/TS			RS + VNS/TS		
w_{max}	BW	%(BW)	RS	%(RS)	Imp.	(%)	CPU (s)	Imp.	(%)	CPU (s)
1	24	1.70	4	0.20	0	0.00	17.88	28	-6.65	16.67
10	5	0.14	25	1.78	25	-4.57	17.57	30	-30.49	17.75
99	6	0.25	23	1.90	29	-4.85	17.94	30	-34.93	17.96
Total	35	2.09	52	3.89	54	-4.72	17.79	88	-24.42	17.46

Table 5: Results for $n = 100$ and $m = 1$

For single machine instances, the performance of the two methods is quasi-identical. Even if RS gives better solutions, improvement is not significant enough to state that this approach is the best. CPU times are comparable.

Inst. type	First place				BW + VNS/TS			RS + VNS/TS		
w_{max}	BW	%(BW)	RS	%(RS)	Imp.	(%)	CPU (s)	Imp.	(%)	CPU (s)
1	27	3.55	2	0.13	0	0.00	8.54	28	-6.83	7.55
10	18	0.99	10	1.01	16	-3.64	8.53	30	-35.43	9.07
99	20	1.19	10	0.63	21	-1.93	9.00	30	-38.71	9.29
Total	65	5.73	22	1.77	37	-2.67	8.69	88	-27.45	8.64

Table 6: Results for $n = 100$ and $m = 3$

We can see here that initial solutions used for the BW variant are improved by only about 3% on average. We are currently investigating whether this is due to the fact that these initial solutions are very good, or whether the VNS/TS needs improvement.

Inst. type	First place				BW + VNS/TS			RS + VNS/TS		
w_{max}	BW	%(BW)	RS	%(RS)	Imp.	(%)	CPU (s)	Imp.	(%)	CPU (s)
1	26	5.23	0	0.00	0	0.00	4.67	26	-10.53	4.61
10	19	2.10	11	1.40	10	-1.66	5.01	30	-42.07	6.29
99	23	2.68	7	0.88	13	-2.76	5.23	30	-45.37	6.36
Total	68	10.01	18	2.28	23	-2.28	4.97	86	-33.69	5.75

Table 7: Results for $n = 100$ and $m = 6$

A general observation is that the inital solution heuristics for BW performs very well, especially on instances where $w_{max} = 1$. With the current settings, no improvements are made by the VNS/TS algorithm on these instances.

4 Conclusion and future directions

In this paper, we have proposed a novel approach that combines variable neighbourhood search and tabu search to solve the parallel machine scheduling problem with the minimisation of the weighted number of late jobs as objective function. We have introduced a solution encoding that takes the specific structure of a solution into account, especially the fact that late jobs do not need to be scheduled. We used this encoding to develop a set of k neighbourhoods, that become larger as the value of k increases. A tabu search algorithm was built, based on these neighbourhoods, and this was embedded in a variable neighbourhood search algorithm, also based on these neigbourhoods.

This approach was compared to an exact method where possible. Although the approach was obviously outperformed in terms of quality, it was shown to be much faster than the exact method. When the exact method failed to find the optimal solution, our approach performed better.

However, the goal of this research was to develop a metaheuristic approach that would be able to solve large instances, where exact methods fail. To the best of our knowledge, there is no other published metaheuristic approach for this specific problem. We have applied our VNS/TS to a number of large instances. Initial tests gives promising results, but also indicates a number of areas for improvement and future research.

References

[1] Ph. Baptiste, A. Jouglet, C. Le Pape, and W. Nuijten. A constraint-based approach to minimize the weighted number of late jobs on parallel machines. Unpublished paper, 2002.

[2] Ph. Baptiste, C. Le Pape, and L. Péridy. Global constraints for partial CSPs: A case study of resource and due-date constraints. In *Proceedings of the 4th international conference on principles and practices of constraint programming*, volume 1520 of *LNCS*, pages 87–120, Pisa, Italy, 1998.

[3] S. Dauzère-Pérès and M. Sevaux. Lagrangean relaxation for minimizing the weigthed number of la te jobs on parallel machines. In *Proceedings of 8th International Workshop on Project Managemen t and Scheduling, PMS'2002*, pages 121–124, Valencia, Spain, 2002.

[4] S. French. *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop.* J.Wiley and Sons, New York, 1982.

[5] M.R. Garey and D.S. Johnson. *Computers and intractability: a guide to theory of $\mathcal{NP}$-completeness.* Freeman, San Franscisco, 1979.

[6] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[7] P. Hansen and N. Mladenović. Variable neighborhood search: principle and applications. *European Journal of Operational Research*, 130:449–467, 2001.

[8] M. Sevaux and P. Thomin. Heuristics and metaheuristics for parallel machine scheduling: a computational evaluation. In *Proceedings of 4th Metaheuristics International Conference, MIC 2001*, pages 411–415, Porto, Portugal, 16-20 July 2001.