Scheduling two interfering job sets on uniform parallel machines with maximum criteria functions

Donatas Elvikis[1], Horst W. Hamacher[1], Vincent T'kindt[2]

[1] University of Kaiserslautern, Germany
elvikis, hamacher@mathematik.uni-kl.de
[2] LI, Université de Tours, France
vincent.tkindt@univ-tours.fr

Keywords: Interfering Jobs, Parallel Machines, Bicriteria Scheduling, Pareto Optimum.

1 Introduction

We consider a m uniform parallel machines scheduling problem of two jobs A and B with n_A and n_B equal processing time operations, respectively. Further, we denote by $n = n_A + n_B$ the total number of operations and by $p_j = \frac{p}{v_j}$, with v_j the processing speed of machine j, the processing time of any operation scheduled on machine j.

Our goal is to minimize two maximum functions associated with jobs A and B and referred to as $F^A_{max} = \max_{i \in A} f^A_i(C_i)$ and $F^B_{max} = \max_{i \in B} f^B_i(C_i)$, respectively, with C_i the completion time of operation i and f_i an increasing function. Therefore we face to a two interfering job sets problem, *i.e.* scheduling each job is done according to its own cost function while sharing common resources with the other job. It is obvious that criteria are conflicting and the problem is referred to as $Q|p_i = p|\#(F^A_{max}, F^B_{max})$ according to the notation presented by T'kindt & Billaut (2006).

As usual when dealing with multiple conflicting criteria, we focus on the calculation of strict Pareto optima for F^A_{max} and F^B_{max} criteria. A schedule σ is a strict Pareto optimum iff there does not exist another schedule $\hat{\sigma}$ such that $F^A_{max}(\hat{\sigma}) \leq F^A_{max}(\sigma)$ and $F^B_{max}(\hat{\sigma}) \leq F^B_{max}(\sigma)$ with at least one strict inequality. Note that each non-dominated criteria vector (F^A_{max}, F^B_{max}) is associated with at least one Pareto optimum. In this paper we focus on the enumeration of such solutions.

Scheduling interfering job sets is a quite recent research area. First, Agnetis, Mirchandani, Pacciarelli & Pacifici (2000) introduce a job shop scheduling problem with two competing players each one having its own optimization goal: this was the starting point for interfering job sets problems. Later, Baker & Smith (2003), Yuan, Shang & Feng (2005) and Agnetis, Mirchandani, Pacciarelli & Pacifici (2004) study several single machine problems with two interfering job sets. The first problem with more than two interfering job sets and unit penalty criteria is dealt with by Cheng, Ng & Yuan (2006) and is shown to be strongly $\mathcal{NP}$-hard. Papers dealing with identical and uniform parallel machines were first published by Balasubramanian, Fowler, Keha & Pfund (2009) and Elvikis, Hamacher & T'kindt (2009), respectively.

2 Solving the $Q|p_i = p|\#(F^A_{max}, F^B_{max})$ problem

It is known that $Q|p_i = p|\#(F^A_{max}, F^B_{max})$ problem can be solved by a naive approach in $\mathcal{O}(n_A^3 n_B + n_A n_B^3)$ time. Also note that efficient algorithms developed by Elvikis et al. (2009) for makespan criterion does not apply since the block property in Lemma 1 does not hold, therefore in this paper we develop new algorithm which enumerates all Pareto solutions of the $Q|p_i = p|\#(F^A_{max}, F^B_{max})$ problem. The algorithm will start with an initial solution where job A is sequenced before B and iteratively look for operations, with costs

equal to the criteria values of the current schedule, to be swapped in order to produce another non-dominated solution.

First observe that as far as criteria on jobs are regular we only need to restrict the search for Pareto optima to the set of active schedules (T'kindt & Billaut 2006). Consequently, and due to the fact that operations are of equal size we can define $\mathcal{T} = \bigcup_{j=1,\dots,m} \{t \mid t = kp_j, k \in \mathbb{N}\}$ as the set of timeslots numbered ascending w.r.t. completion times. Thus $\mathcal{T} = \{t_1 \leq t_2 \leq \cdots \leq t_n\}$ is the set of timeslots for any active schedule σ and minimizing criteria F^A_{max} and F^B_{max} is equivalent to assign operations to their completion times. In other words, the uniform parallel machines problem can be seen as a particular single machine problem for which the sequence of completion times is fixed and we only need to assign jobs to timeslots which makes it more particular than the problems tackled by Agnetis et al. (2004).

Further we give definitions of the additional solution properties which we use throughout the paper.

Definition 1. *Schedule $\hat{\sigma}$ such that $\hat{F}^A_{max} > F^A_{max}$ and $\hat{F}^B_{max} < F^B_{max}$ holds and there is no other schedule $\tilde{\sigma}$ with $\hat{F}^A_{max} > \tilde{F}^A_{max} > F^A_{max}$ and $\hat{F}^B_{max} < \tilde{F}^B_{max} < F^B_{max}$ is called a* ***succeeding*** *solution of σ.*

Definition 2. *Solution σ such that $f^A_{\sigma(k)}(t_r) > F^A_{max}$, $r = \arg\min_{r=k+1,\dots,n}\{\sigma(r) \in B\}$, holds for all operations $\sigma(k) \in A$, with $k = 1, \dots, n$, is called* ***saturated****.*

In other words, if any operation of A is right time shifted in σ, then it will issue a greater criterion F^A_{max} value. Consequently, saturated solution provides job B with the timeslots aligned maximally to the left, *i.e.* with the earliest possible completion times.

We start the algorithm with a saturated solution σ minimizing $Lex(F^A_{max}, F^B_{max})$ which can be found by adoption of the algorithm described by Agnetis et al. (2004) for solving $1||\varepsilon(F^A_{max}/F^B_{max})$. Let F^B_{max} be given by operation $u \in B$ sequenced in position ℓ in σ and ties are broken in favor of the maximal ℓ. Obviously in succeeding schedule $\hat{\sigma}$ operation u must be sequenced at some earlier position $r < \ell$ such that $t_r < t_\ell$.

Lemma 1. *Given two succeeding non-dominated schedules σ and $\hat{\sigma}$. Let the F^B_{max} criterion value for σ be due to the operation $u \in B$ sequenced in position ℓ in σ. Then $\exists v \in A$ such that $C^A_v(\sigma) < t_\ell$ and $C^A_v(\hat{\sigma}) \geq t_\ell$.*

Lemma 2. *Let $v \in A$ be such that $C^A_v(\sigma) < t_\ell$ and $C^A_v(\hat{\sigma}) \geq t_\ell$ where σ and $\hat{\sigma}$ are two succeeding non-dominated solutions and t_ℓ such that $f^B_{\sigma(\ell)}(t_\ell) = F^B_{max}$ with $\sigma(\ell) \in B$. Then $f^A_v(C^A_v(\hat{\sigma})) = \hat{F}^A_{max}$ holds.*

Let τ_g be the time of the g-th smallest intersection point between two f^B_i functions, *i.e.* $\tau_1 \leq \tau_2 \leq ... \leq \tau_M$ with M the total number of intersection points. Notice that if two functions $f^B_{i'}$ and $f^B_{i''}$ overlap in some region, thus have an infinite number of intersection points, we only consider the point with the highest value.

Further in the paper we use the notion of *Ordered Operation Sequence* (OOS), defined by $\mathcal{S}_g = \{\mathsf{s}_g(1), \dots, \mathsf{s}_g(n_B)\}$: such permutation of job B operations that $f^B_{\mathsf{s}_g(i)}(t) \leq f^B_{\mathsf{s}_g(i+1)}(t)$, $\forall t \in [\tau_{g-1}; \tau_g[$ with $\tau_0 = 0$ holds. Besides, we refer to $\mathcal{T}_g$ as the set of timeslots for which the sequence $\mathcal{S}_g$ does not change, *i.e.* $\mathcal{T}_g = \{t \in \mathcal{T} \mid t \in [\tau_{g-1}; \tau_g[\}$. It is important to notice that we only consider at most n sequences S_g such that $\mathcal{T}_g \neq \emptyset$. Without loss of generality in the remainder we assume that all S_g's with $\mathcal{T}_g \neq \emptyset$ are numbered consecutively starting from index 1. At last, we define $\mathcal{L}_t(\sigma) \subseteq \mathcal{S}_g$ with $t \in \mathcal{T}_g$ as *unscheduled OOS* (uOOS), the permutation of yet unscheduled B operations regarding a backward sequence σ.

Observe that both OOS's can be computed while building an initial non-dominated solution without added time complexity since we only need to compute those valid for timeslots t_k. Moreover, $\mathcal{S}_g$ and $\mathcal{T}_g$ do not depend on the actual schedule, so they are valid for any solution σ. On the other hand we have to manage n uOOS lists $\mathcal{L}_t(\sigma)$ which can be done in an efficient way as we will see later on.

Now we turn to the algorithmic side of the section and describe how succeeding solutions are found. Let σ be a saturated strictly non-dominated schedule as described in Definition 2. Then due to Lemma 1 there exists operation $v \in A$ in position k in σ which will be moved to a later position $\ell > k$ in the new schedule $\hat{\sigma}$, here ℓ is the position of operation $u \in B$ such that $f_u^B(C_u^B(\sigma)) = F_{max}^B(\sigma)$ and ℓ is maximal. Note that both operations u and v with their respective positions ℓ and k can be found in $O(n)$ time by simple scan of the schedule. Moreover, operation u is not sequenced in $\hat{\sigma}$ yet, thus in the following we will describe a procedure which fills in the timeslot t_k and schedules operation u in $\mathcal{O}(n_B \log n_B)$ time. Here we want to point out that this is not the final step needed to produce a saturated non-dominated solution which answers conditions defined in Definition 2, but it is crucial in reaching one.

First note that none of the operations in B sequenced in positions after ℓ can fill the vacant position k in $\hat{\sigma}$ since otherwise $v \in A$ would be postponed more than necessary and due to Lemma 2 increase criteria $\hat{F}_{max}^A$ value. Thus only $1, \ldots, \ell-1$ timeslots in $\hat{\sigma}$ have to be considered in the rescheduling process. Moreover, we do not consider other operations in A than v which has already been shifted to position ℓ in $\hat{\sigma}$. This implies that we only need to reschedule some timeslots from t_1 to $t_{\ell-1}$ in which operations of job B in σ were scheduled. We consider them in reverse order and split into the following two cases.

Case 1 ($t_{\ell-1}, \ldots, t_{k+1}$). This case is quite trivial since for each timeslot t_r, $\forall r = \ell-1, ..., k+1$, the assigned operation i in $\hat{\sigma}$ has the lowest cost value $f_i^B(t_r)$ among operations in the associated uOOS sequence $\mathcal{L}_{t_r}(\hat{\sigma})$. Henceforth, we only need to compare such operation i with operation u to decide which one will be assigned to the timeslot t_r in $\hat{\sigma}$: either operation i stays at position r if $f_i^B(t_r) < f_u^B(t_r)$ or u is sequenced at position r and i becomes unscheduled. Consequently, operation u may be updated to i as well as the sequences $\mathcal{L}_t(\hat{\sigma})$, $\forall t \leq t_r$. This can be implemented in constant time if we use Brodal (1996) priority queue for $\mathcal{L}_t(\sigma)$.

Case 2 ($t_k, \ldots, t_1$). Assume that $t_k \in \mathcal{T}_g$ and it has the smallest completion time among all $t \in \mathcal{T}_g$. If this is not the case, then due to the fact that $\mathcal{S}_g$ is the same for all $t \in \mathcal{T}_g$ we can proceed similarly as in Case 1 and do pairwise comparison of cost functions between operations i sequenced in position $k-1$ in $\hat{\sigma}$ and u with respect to the timeslot t_k. This is repeated until: *(i)* we reach timeslot t_k which satisfies the assumption above and continue with this case; *(ii)* find such t_k that $f_u^B(t_k) \leq f_i^B(t_k)$ and sequence operation u to position k in $\hat{\sigma}$, thus procedure is completed or *(iii)* reach $k = 1$.

Under assumption above we have two operations that can be assigned to timeslot t_k, namely operation u and the first unscheduled operation i in $\mathcal{L}_{t_k}(\sigma)$. Note that $u \notin \mathcal{L}_{t_k}(\sigma)$, thus adding it to $\mathcal{L}_{t_k}(\sigma)$ and sequencing first operation from the uOOS list, *i.e.* $\mathcal{L}_{t_k}(\sigma)[1]$, gives an optimal assignment to position k in the new schedule $\hat{\sigma}$. Moreover, if $\mathcal{L}_{t_k}(\sigma)[1]$ is u, then procedure is completed, otherwise we have to update ℓ to the value of k and k to q, the position where i was sequenced in σ, and start again with the Case 1. Note that adding operation u to the uOOS takes constant time, whereas removing the minimum element $\mathcal{L}_{t_k}(\sigma)[1]$ takes $O(\log n_B)$ time. Moreover, if $i \neq u$ then all $\mathcal{L}_{t_r}(\sigma)$ with $r = q+1, \ldots, k-1$ have to be updated by deleting operation i. Removing an arbitrary element from Brodal (1996) priority queue can be implemented in logarithmic time.

Note that these two basic steps create a succeeding feasible schedule $\hat{\sigma}$, however this one does not necessarily satisfy conditions in Definition 2. Hence we saturate solution $\hat{\sigma}$ by

postponing all operations of A until conditions in Definition 2 hold. Observe that this can be done iteratively by identifying operation $v \in A$ in position k such that $f^A_{\hat{\sigma}(k)}(t_r) \leq \hat{F}^A_{max}$, with $r = \arg\min_{r=k+1,\ldots,n}\{\sigma(r) \in B\}$, and then applying procedure above for rescheduling operations in B. Operation v can be found in $O(n)$ time by scanning schedule $\hat{\sigma}$. Moreover, Agnetis et al. (2004) showed in Theorem 11.3 that for the $1||\#(F^A_{max}, F^B_{max})$ problem there are at most $n_A n_B$ iterations when some A operation i can be postponed in favor of an earlier position for operation j in B and once j precedes i in σ there is no succeeding schedule $\hat{\sigma}$ with i and j order reversed. Note that these results can be easily transported for the $Q|p_i = p|\#(F^A_{max}, F^B_{max})$ problem.

Finally we have transformed the starting non-dominated solution σ to the next succeeding non-dominated solution $\hat{\sigma}$ which can be used as an initial solution for further iterations of the algorithm.

As outlined at the beginning we know that the initial saturated lexicographical solution σ can be found in $\mathcal{O}(n_A^2 + n_B^2)$ time, whereas each iteration of the algorithm takes $\mathcal{O}((n_A + n_B)\log n_B)$ time and there are at most $\mathcal{O}(n_A n_B)$ of them, thus the time needed to enumerate all non-dominated solutions of the $Q|p_i = p|\#(F^A_{max}, F^B_{max})$ problem is $\mathcal{O}((n_A^2 n_B + n_A n_B^2)\log n_B)$. Observe that due to the necessity of unscheduled ordered operation lists for each timeslot space complexity is quadratic to the size of the problem.

Theorem 1. *Algorithm solves $Q|p_i = p|\#(F^A_{max}, F^B_{max})$ in $\mathcal{O}((n_A^2 n_B + n_A n_B^2)\log n_B)$ time with $\mathcal{O}(n_A n_B + n_B^2)$ memory.*

Acknowledgements

The research has been partially supported by the program "Center for Mathematical and Computational Modelling $(CM)^2$".

References

Agnetis, A., Mirchandani, P., Pacciarelli, D. & Pacifici, A. (2000), 'Nondominated schedules for a job-shop with two competing users', *Computational & Mathematical Organization Theory* **6**(2), 191–217.

Agnetis, A., Mirchandani, P., Pacciarelli, D. & Pacifici, A. (2004), 'Scheduling problems with two competing agents', *Operations Research* **42**(2), 229–242.

Baker, K. & Smith, J. (2003), 'A multiple-criterion model for machine scheduling', *Journal of Scheduling* **6**, 7–16.

Balasubramanian, H., Fowler, J., Keha, A. & Pfund, M. (2009), 'Scheduling interfering job sets on parallel machines', *European Journal of Operational Research* **199**(1), 55 – 67.

Brodal, G. S. (1996), Worst-case efficient priority queues, *in* 'SODA '96: Proceedings of the seventh annual ACM-SIAM symposium on Discrete algorithms', Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pp. 52–58.

Cheng, T., Ng, C. & Yuan, J. (2006), 'Multi-agent scheduling on a single machine to minimize total weighted number of tardy jobs', *Theoretical Computer Science* **362**, 273–281.

Elvikis, D., Hamacher, H. W. & T'kindt, V. (2009), Scheduling two interfering job sets on uniform parallel machines with makespan and cost functions, *in* 'Proceedings of the fourth Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2009)', pp. 645–654.

T'kindt, V. & Billaut, J.-C. (2006), *Multicriteria Scheduling: Theory, Models and Algorithms*, 2nd edition, Springer (Berlin).

Yuan, J., Shang, W. & Feng, Q. (2005), 'A note on the scheduling with two families of jobs', *Journal of Scheduling* **8**, 537–542.