Search on the Enumeration Tree in the Multiprocessor Job-Shop Problem Shop

Lester Carballo* Alexander Lazarev** Nodari Vakhania*
Frank Werner***

* Science Faculty, UAEM, Mexico

** Institute of Control Sciences, Russian Academy of Sciences, Russia

*** Faculty of Mathematics, Otto-von-Guericke-University Magdeburg, Germany

INCOM 2012, Bucharest/Romania, May 23-25, 2012

The problem formulation

$JR|prec|C_{max}$, $JQ|prec|C_{max}$ and $JP|prec|C_{max}$ (MJSP) are extensions of the classical **job-shop scheduling problem** (JSP) $J||C_{max}$ (we have an arbitrary task graph and parallel alternative processors).

We have **tasks (operations)** from $\mathcal{O} = \{1, 2, ..., n\}$ and m different processor (machine) groups, $\mathcal{M}_k$ being the kth group of parallel **processors (machines)**, and P_{kl} is the lth processor of this group.

Each $o \in O_k$ is to be performed by any processor of $\mathcal{M}_k$. d_{iP} is the (uninterrupted) processing time of task i on processor P. Each group of parallel processors can be **unrelated**, **uniform** or **identical**.

The constraints and schedules

The **resource constraints**: For each two tasks i, j such that $P(i) = P(j) = P$, either $s_i + d_{iP} \leq s_j$ or $s_j + d_{jP} \leq s_i$ should hold, where s_i is the starting time of i and $P(i)$ is the processor to which task i is assigned.

The **precedence constraints**: for each $i \in \mathcal{O}$ we are given the set of immediate predecessors $pred(i)$ of task i, task i becomes **ready** when all tasks from $pred(i)$ are finished.

A **schedule (solution)** assigns to each task a particular processor and a starting time (on that processor); a **feasible schedule** is a schedule satisfying the constraints. An **optimal schedule** is a feasible schedule which minimizes the **makespan**, i.e., the maximal task completion time.

Some problem characteristics

If in an instance of MJSP from each group of processors all processors except an arbitrarily selected one are eliminated, then a corresponding instance of JSP is obtained.

$JR|prec|C_{max}$ is reducible to $R|prec|C_{max}$: For each $o \in O_k$, $k = 1, \dots, m$, we let $d_{oM} = \infty$ if $M \notin \mathcal{M}_k$. $JR|prec|C_{max}$ can be also seen as a **resource-constrained project scheduling problem**: associate the kth machine group with the kth resource which amount is the number of parallel machines in the group. The requirement of the kth resource of each operation from O_k is 1 and that of any other operation is 0.

JSP is (strongly) NP-hard with no approximation algorithms with a guaranteed performance. To find an optimal (near-optimal) schedule we need to enumerate an exponentially growing number of schedules (to work with a large subsets of schedules).

Some related work

There are numerous enumerative algorithms for JSP which apply **lower bounds** to reduce the number of active schedules to be considered. Some of them are given by Adams, Balas and Zawack 1988, Carlier and Pinson 1989, Lageweg, Lenstra and Rinnooy Kan 1977 and McMahon and Florian 1975.

There are few enumerative algorithms for our extensions. Some of them are given by Giffer and Thompson 1960, Carlier and Pinson 1998 (identical processors) and Vakhania and Shchepin 2002 (unrelated processors).

Active schedules

An optimal schedule is among **active schedules** (in an active schedule no task can start earlier than it is scheduled without delaying some other task). We try to reduce the set of active schedules to be considered on the base of some local dominance relations, before using (expensive) lower bounds.

The feasible solution space of $JR|prec|C_{max}$ is significantly larger than that of JSP: n simultaneously available operations can be assigned to m machines in $(n+m-1)!/(m-1)!$ different ways, while the corresponding number for a single machine is $n!$.

A quick dominant set

A **quick processor** is a fastest one for a given task at a given stage (a quick processor for a given task can be found in time, linear to the number of processors in the corresponding group).

A set of ready tasks, conflicting on a machine, which is quick for at least one of these operations, is a **quick set**. We are allowed to branch by a quick set, if it is **dominant**. Intuitively, if we branch by a dominant set, then we are guaranteed that we will not delay any not yet ready task (not included in the set) from the same group.

Our earlier work, the first step

We accomplish a two-step preliminary reduction of the set of active schedules for MJSP using **local estimations** (Vakhania & Shchepin 2002). With the first step, we always branch by a single quick dominant set; i.e., all parallel machines are discarded except the selected quick machine. This already guarantees that the number of the generated solutions for a given instance of MJSP is no more than that for any corresponding instance of JSP.

The second step

In the second step, an “artificial” relaxation of conflicts between the operations of the conflict sets is carried out: the branching by some tasks is postponed whenever this is possible. The selected quick dominant set is partitioned into specially determined subsets. To each subset its own processor, which is quick for at least one task from this subset, corresponds. Then, instead of branching by the quick set, branchings are performed by the subsets from the partition, on different processors on different levels of the solution tree. So the concurrent jobs from different subsets are processed in parallel.

The efficiency of the reduction

With a probability of almost 1, the number of generated solutions, as compared to the number of all active schedules, **decreases exponentially** with the number of processors and tasks in each group of machines and operations, as follows. If we let ν and μ to be the number of operations and machines in each subset of operations and machines, then with a probability of almost 1, the algorithm generates approximately $(\mu)^{m\nu}$ and $2^{m(\mu-1)}\mu^{m\nu}$ times less feasible schedules than the number of all active feasible schedules of any corresponding instance of JSP and our generalized problem, respectively.

Some terminology

The (compact) solution tree T represents all generated (partial and complete) solutions. With an edge, incident out from a node $h \in T$ (a **stage**), a single task from a bunch of ready tasks (the **branching set**) is associated. The branching set $\mathcal{C}_h$ of stage h is characterized by a particular processor on which the operations of this set are to be actually scheduled on stage h. By the branching, the resource conflicts in $\mathcal{C}_h$ are resolved (each alternative processor implies its own conflicts).

In this way there are $|\mathcal{C}_h|$ posssible extensions of the current (partial) schedule σ_h [a (partial) permutation of n tasks]. $\sigma_h i^P$ is an extension of σ_h with task i scheduled on processor P at stage h.

σ_h is represented by a directed weighted graph G_h. The digraph $G_0 = (X, E_0)$ is associated with the root of T. To each task $i \in O$ corresponds the unique node $i \in X$. There is one fictitious initial node 0, preceding all nodes, and one fictitious terminal node $n+1$, succeeding all nodes in G_0.

E_0 is the arc set consisting of the arcs (i, j), for each task i, directly preceding task j; $(0, i) \in E_0$ if task i has no predecessors and $(j, n+1) \in E_0$ if task j has no successors.

$w(i, j)$ is the weight associated with $(i, j) \in E_0$; initially, assign to $w(i, j)$ the minimal processing time of task i, later we correct these weights when we assign a task to the particular processor.

Let (h, h') be an edge in T with the associated task j assigned to processor P. $G_{\sigma_{h'}}$ is obtained from G_{σ_h} by completing the arc set of the latter graph with the arcs of the form (i, j), with the associated weights $w(i, j) = d_{iP}$, for each task i, scheduled earlier on the processor P. We correct the weights of all arcs incident out from node j $(j, o) \in E_0$, as $w(j, o) := d_{jP}$.

The length of a critical path in $G_{h'}$ is the makespan $|\sigma_{h'}|$ of $\sigma_{h'} = \sigma_h j^P$.

General points

If a lower bound $L(\sigma_h)$ of the partial solution σ_h is more than or equal to the makespan $|\sigma|$ of some already generated complete solution σ, then all extensions of σ_h can be abandoned. $L(\sigma_h)$ cannot be greater than the makespan of the best potential extension of σ_h (since then we could loose this extension), but it should be as close as possible to this value (because then the more are the chances that $L(\sigma_h) \geq |\sigma|$).

Some notations and trivial bounds

Consider the set of (yet unscheduled) tasks $\mathcal{O}_{kh}$. Let $o \in \mathcal{O}_{kh}$. The length of a longest path to node o in G_h is the earliest possible starting (release) time or the **(head)** $\mathrm{head}_h(o)$ of task o by stage h.

$\mathrm{tail}_h(o)$, the **tail** of o at stage h, is the critical path length from note o to the sink node of G_h (processing time of o is not counted).

A trivial lower bound $L_T(\sigma_h o^Q) = \tau(\sigma_h) + \mathrm{tail}_h(o)$ ignores all yet unresolved potential conflicts (the processing times of yet unscheduled tasks). A stronger lower bound would take into account a possible contribution of the yet unscheduled tasks, this will involve some optimal scheduling on parallel machines.

Relaxing all resource constrains except the ones of $\mathcal{M}_k$

Relax (ignore) at stage h all resource constraints except the ones of some $\mathcal{M}_k$: each $o \in \mathcal{O}_{kh}$ cannot be started earlier than at time $\mathrm{head}_h(o)$, and once it is completed, it will take at least $\mathrm{tail}_h(o)$ time for all successors of o to be finished. o can be scheduled on any of the machines of $\mathcal{M}_k$ having a processing time d_{iP} on machine $P \in \mathcal{M}_k$. Each $P \in \mathcal{M}_k$ has its release time $R_h(P)$ (the completion time of the task, scheduled last by stage h on P).

The auxiliary problem $\mathcal{A}_{kh}$ is that of scheduling tasks with release times and tails on a group of parallel machines $\mathcal{M}_k$ with the objective to minimize the makespan. An optimal ordering of operations of $\mathcal{O}_{kh}$ on machines from $\mathcal{M}_k$ is a lower bound for MJSP.

A bottleneck machine-group

A **bottleneck** machine group is a one which results the maximal makespan among all yet unscheduled machine groups $\mathcal{M}_k$. We may find all m lower bounds for node h and take the maximum which will be a lower bound for MJSP.

A similar relaxation approach has been used for JSP, for example Adams, Balas and Zawack 1988, Carlier and Pinson 1989, Lageweg, Lenstra and Rinnooy Kan 1977. Instead of dealing with $1|r_i, q_i|C_{max}$ in case of JSP, now we deal with $R/Q/P|r_i, q_i|C_{max}$.

The time complexity of the derived scheduling problems

$P|r_i, q_i|C_{max}$, $Q|r_i, q_i|C_{max}$, $R|r_i, q_i|C_{max}$ are all NP-hard. Even $1|r_i, q_i|C_{max}$ is NP-hard, but there are exponential algorithms with a good practical behavior for this problem and it has been used in branch-and-bound algorithms for JSP as a one-machine relaxation (McMahon & Florian 1975 and Carlier 1981).

Unfortunately, there are no known algorithms with good practical performance for $P|r_i, q_i|C_{max}$ (the version with identical machines) and so much for $R|r_i, q_i|C_{max}$. We need to find some other way around.

Simple lower bounds based on earlier existing algorithms

Carlier & Pinson 1989 have suggested a lower bound for $JP|prec|C_{max}$. They proposed an $O(n \log n + nm \log m)$ algorithm for the non-sequential version of $P|r_i, q_i, pmtn|C_{max}$ which is a lower estimation of the optimal makespan for $P|r_i, q_i, pmtn|C_{max}$. At the expense of weakening the bound, the solution of the above problem can be used as a lower bound for MJSP:

Let $d_o^{min} = min\{d_{oM}, M \in \mathcal{M}_k\}$. We replace the unrelated machine group $\mathcal{M}_k$ with the identical machine group $\mathcal{M}_k^{'}$, defined as follows: the number of machines in both groups is the same, and for each $o \in O_k$ and $M \in \mathcal{M}_k^{'}$, $d_{oM} = d_o^{min}$. An optimal solution of the obtained instance of $P|r_i, q_i, pmtn|C_{max}$ with $\mathcal{M}_k^{'}$ is no more than that of the corresponding instance of $R|r_i, q_i, pmtn|C_{max}$ with $\mathcal{M}_k$.

Is it possible to find a better "approximation" with an identical machine group of the unrelated machine group $\mathcal{M}_k$, i.e., to increase d_{oM}, $o \in \mathcal{O}_k$, $M \in \mathcal{M}_k^{'}$?

For uniform machines, we can obtain a stronger lower bound by using the algorithm of Federgruen & Groenevelt 1986 for $Qm|r_i, q_i, pmtn|C_{max}$ with the time complexity of $O(tn^3)$ (t is the number of machines with distinct speeds).

As to $JR|prmt|C_{max}$, the technique based on linear programming of Lawler & Labetoulle 1978 yields a polynomial-time algorithm for $Rm|r_i, q_i, pmtn|C_{max}$. This is a lower estimation of the optimal makespan for $Rm|r_i, q_i|C_{max}$ which, in turn, provides a lower bound for $JR|prmt|C_{max}$.

Further bounds and computational results

We have also developed some alternative methods to obtain more sophisticated lower bounds.

We have tested the generated C++ code for the reduction algorithm (without the lower bounds) on randomly generated instances of moderate sizes (up to 20 operations on 5 groups of parallel machines).

The reduction of the solution space was between 15 and 25 % compared to the number of active schedules.

A more essential reduction is expected for larger instances when also lower bounds will be incorporated into the code (what is planned as future work)