Heuristic Algorithms for a Job-Shop Problem with Minimizing Total Job Tardiness

Yuri N. Sotskov[1], **Omid Gholami**[1], **Frank Werner**[2]

[1] *United Institute of Informatics Problems, 220012 Minsk, Belarus;* sotskov@newman.bas-net.by, gholami_iran@yahoo.com

[2] *Fakultät für Mathematik, Otto-von-Guericke-Universität Magdeburg, PSF 4120, 39016 Magdeburg, Germany;* frank.werner@ovgu.de

In practice, it is often required to process a set of jobs without operation preemptions satisfying temporal and resource constraints. Temporal constraints say that some jobs have to be finished before some others can be started. Resource constraints say that operations processed on the same machine cannot be processed simultaneously. The objective is to construct a schedule specifying when each operation starts such that both temporal and resource constraints are satisfied and the given objective function has a minimum value. One can model such a scheduling process via the following job-shop problem.

There are n jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ to be processed on m machines $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$. Operation preemptions are not allowed, and the machine routes $\mathcal{O}^i = (O_{i1}, O_{i2}, \ldots, O_{in_i})$ for processing the jobs $J_i \in \mathcal{J}$ may be given differently for different jobs. The time $p_{ij} > 0$ needed for processing an operation O_{ij} of a job J_i on the corresponding machine $M_v \in \mathcal{M}$ is known before scheduling. A job $J_i \in \mathcal{J}$ is available for processing from time-point $r_i \geq 0$. The time-point $d_i > r_i$ defines a due date for completing job J_i. It is assumed that machine $M_k \in \mathcal{M}$ can process a job $J_i \in \mathcal{J}$ at most once. Consequently, any two operations O_{ij} and O_{ik}, $j \neq k$, of the same job $J_i \in \mathcal{J}$ have to be processed by different machines and $n_i \leq m$ (such a scheduling problem is called a classical job-shop). We consider the objective of finding a schedule minimizing the sum $\sum_{i=1}^{n} T_i$ of the tardiness times $T_i = \max\{0, C_i - d_i\}$ for the jobs $J_i \in \mathcal{J}$. Hereafter, C_i denotes the completion time of a job $J_i \in \mathcal{J}$. According to the three-field notation $\alpha|\beta|\gamma$ used for machine scheduling problems, the above problem is denoted as $J|r_i|\sum T_i$.

Problem $J|r_i|\sum T_i$ arises, e.g., in train scheduling for a single-track railway network: To determine a schedule for a set of trains that does not violate the single-track capacities and the train timetable. In a single-

track railway, a pair of sequential stations can be connected by a single-track (railroad section) only. Specifically, this is the case for most railway networks in countries of the Middle East.

In a job-shop approach to train scheduling, trains and railroad sections are synonymous with the jobs $\mathcal{J}$ and the machines $\mathcal{M}$, respectively. An operation O_{ij} is regarded as a movement of the train $J_i \in \mathcal{J}$ across the railroad section $M_v \in \mathcal{M}$, where machine M_v has to process operation O_{ij}. The positive number p_{ij} denotes the time required for train $J_i \in \mathcal{J}$ to travel through section $M_v \in \mathcal{M}$. The non-negative number r_i denotes the earliest possible departure time for the train (release time of the job) $J_i \in \mathcal{J}$ from the original station in the route $\mathcal{O}^i$. The positive number d_i denotes the official arrival time of the train (due date for completing the job) $J_i \in \mathcal{J}$ at the terminal station in the route $\mathcal{O}^i$. It should be noted that for train scheduling, the inequality $m > n$ holds and each machine $M_k \in \mathcal{M}$ can process a job $J_i \in \mathcal{J}$ at most once.

Our aim was to find an algorithm for the problem $J|r_i|\sum T_i$ to be fast even for a large size of the input data (this is the case for a real-world railway scheduling problem). It is clear that an exact branch and bound method creates a lot of branches in the solution tree for a large input data, and so it is not possible to use a branch and bound method for most real-world job-shop problems with large sizes. Heuristic methods like a genetic algorithm are basically rather slow. Furthermore, using algorithms like Lagrangian relaxation or simulated annealing can reduce the computational time only a bit. A lot of methods like tabu search need many calculations and cannot satisfy our aim as well.

Problem $J|r_i|\sum T_i$ is very complicated in the computational sense since even its special cases belong to the class of binary (or unary) NP-hard problems [1]. In order to achieve a practical size of a classical job-shop problem, which can be solved heuristically within a reasonable time, we first coded a shifting bottleneck algorithm, which was originated in [2] for a job-shop problem $J||C_{max}$ with the makespan criterion $C_{max} = \max\{C_i : J_i \in \mathcal{J}\}$. We tested the program realizing the shifting bottleneck algorithm for the problem $J|r_i|\sum T_i$ as one of the most famous heuristic algorithms for job-shop problems [2] (this algorithm was improved in [3]). However, we obtained unsatisfactory large CPU-times for randomly generated instances $J|r_i|\sum T_i$ with large and even moderate numbers m of machines provided that $m > n$. Therefore, we were forced to look for other heuristic algorithms for the problem $J|r_i|\sum T_i$, which

will run essentially faster than the shifting bottleneck algorithm and will provide sufficiently close objective function values when $m > n$ and each machine $M_v \in \mathcal{M}$ may process at most one operation from the route $\mathcal{O}^i$ of a job $J_i \in \mathcal{J}$.

We observed that many recursive functions are needed to calculate data like a local due date, a critical path and a local release time for each vertex (i.e., operation) in the mixed graph $G = (Q, A, E)$ representing a job-shop [4]. In the mixed graph $G = (Q, A, E)$, the vertex set Q is the set of all operations including a source operation O and a sink operation O_i for each job $J_i \in \mathcal{J}$. The arc set A defines temporal constraints given by the routes $\mathcal{O}^i$ of the jobs $J_i \in \mathcal{J}$. The edge set E defines resource constraints given by the machine set $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$ and by the routes $\mathcal{O}^i$ for the jobs $J_i \in \mathcal{J}$ (see [4] for details).

The earliest start time r_{ij} of an operation $O_{ij} \in Q$ may be defined as the length of the longest path from a start vertex $O \in Q$ to vertex O_{ij} in the digraph $(Q, A, \emptyset)$ obtained from the mixed graph G via deleting all the edges E. We call r_{i1} the release time of a job J_i provided that $O_{i1} \in Q$ is the first operation of a job J_i in the route $\mathcal{O}^i$. Since the feasible digraph $(Q, A, \emptyset)$ has no circuits, all the earliest start times r_{i1}, $i \in \{1, 2, \ldots, n\}$, are finite and can be calculated in linear time of the sum $|Q| + |A|$.

We focus on the shortest release times of the operations Q in the following algorithm which is called SRT-algorithm. The shortest release time of an operation is used as a priority (SRT-priority) in the SRT-algorithm. In contrast to the shifting bottleneck algorithm, which examines a bottleneck machine at each iteration [2,3], a critical job is examined at each iteration of the SRT-algorithm.

In the initial step of the SRT-algorithm, the earliest start times r_{ij} of all operations $O_{ij} \in Q$ have to be computed due to the following recursion: $r_{ij} = r_{i,j-1} + p_{i,j-1}$. The release time of the source operation O is equal to $\min\{r_i : J_i \in \mathcal{J}\}$.

The first job to be examined is a job $J_i \in \mathcal{J}$, whose last operation O_{in_i} in the route $\mathcal{O}^i$ is the next to the last one in the critical path of the digraph $(Q, A, \emptyset)$. At the first iteration of the SRT-algorithm, the following two steps are realized.

Step 1. The SRT-algorithm finds the first request (i.e., operation $O_{i1} \in Q$) of job J_i for the machine $M_v \in \mathcal{M}$ processing operation O_{i1}. Then the algorithm compares the release time r_{i1} with the release times of all operations $O_{jg} \in Q$ of the other jobs on the same machine $M_v \in \mathcal{M}$

processing operation O_{i1}. To resolve conflicts of jobs for the same machine, the SRT-priority is used as follows. If the release time r_{i1} is not greater than that of operation O_{jg} for the same machine $M_v \in \mathcal{M}$, then the arc (O_{i1}, O_{jg}) starting from operation O_{i1} and ending in operation O_{jg} has to be added to the digraph $(Q, A, \emptyset)$. Otherwise, the symmetric arc (O_{jg}, O_{i1}) has to be added to the digraph $(Q, A, \emptyset)$. If an arc is added to the digraph $(Q, A, \emptyset)$, some local release times may be changed after the second step of the SRT-algorithm.

Step 2. The release time of the destination vertex of an arc and the other vertices related to this vertex must be checked and the corresponding release times must be modified if it is necessary. A job priority will be defined by an arc between two operations requesting the same machine, one is the starting vertex of the arc (let this vertex be O_{km}) and the other one is an end vertex of the arc (let it be O_{ij}). A new release time of the operation O_{ij} has to be calculated as follows: $r_{ij} := \max\{r_{ij}, r_{km} + p_{km}\}$, where the maximum has to be taken over all arcs (O_{km}, O_{ij}) belonging to the digraph already constructed. The above equation must be recursively applied to each vertex of the digraph that has an incoming arc from the vertex O_{ij} until the sink vertex O_i. If the release time was not changed, the calculation of the recursive function is stopped.

Steps 1 and 2 are repeated for operation O_{i2}, then for operation O_{i3}, and so on until operation O_{in_i} of the route $\mathcal{O}^i$. As a result, the mixed graph G is transformed into a mixed graph denoted as $G_i = (Q, A \bigcup A_i, E \setminus E_i)$.

The second job to be examined is the "second critical" job, i.e., a job $J_u \in \mathcal{J} \setminus \{J_i\}$, whose last operation in the route $\mathcal{O}^u$ has the largest completion time in the digraph $(Q, A \bigcup A_i, \emptyset)$ among all jobs from the set $\mathcal{J} \setminus \{J_i\}$. So, at the second iteration, steps 1 and 2 are executed for the job J_u, the digraph $(Q, A \bigcup A_i, \emptyset)$, and the mixed graph G_i.

The process is continued for the "third critical" job, then for the "fourth critical" job and so on, until all jobs $\mathcal{J}$ have been accounted. As a result, the mixed graph G is transformed into the digraph $G_h = (Q, A \bigcup A_h, E \setminus E_h)$, where $E \setminus E_h = \emptyset$ and job J_h was examined at the h-th iteration. It is easy to convince that using the SRT-priority cannot generate a circuit in the digraphs $(Q, A \bigcup A_i, \emptyset)$, ..., $(Q, A \bigcup A_h, \emptyset)$ constructed from the first to the last iterations. A circuit-free digraph G_h uniquely determines a semiactive schedule [4], which may be built via the critical path method in $O(n + |A_h|)$ time.

Both the shifting bottleneck algorithm and the SRT-algorithm have been coded in Delphi and tested on a laptop computer. We compared the CPU-time taken by the SRT-algorithm and by the shifting bottleneck algorithm to solve the same randomly generated problems $J|r_i|\sum T_i$ heuristically for different values $n \leq 20$ and $m \leq 20$. The SRT-algorithm runs considerably faster than the shifting bottleneck algorithm if $m > n$. In the experiments, we compared the objective function values for randomly generated problems $J|r_i|\sum T_i$ with different products $n \times m$.

The computational results showed that there is no meaningful difference between the quality of the schedules obtained by the two algorithms tested for randomly generated problems $J|r_i|\sum T_i$. The computational experiments also evaluated the effect of adding either new jobs or new machines to the CPU-time required to solve randomly generated problems $J|r_i|\sum T_i$. When we increased the number of jobs (machines, respectively) of the randomly generated instances $J|r_i|\sum T_i$, the CPU-time needed for the SRT-algorithm increased considerably (very slowly).

The SRT-algorithm used a smaller CPU-time than the shifting bottleneck algorithm especially when the number of machines m was essentially larger than the number of jobs n. Since the total tardiness values $\sum T_i$ of the schedules constructed by the shifting bottleneck algorithm and the SRT-algorithm are rather close, we can claim that the SRT-algorithm is a good heuristic for a job-shop problem $J|r_i|\sum T_i$ with large $m > n$.

As observed from computational experiments, the SRT-algorithm is a good choice for the classical job-shop problem $J|r_i|\sum T_i$ when the number of machines is much larger than the number of jobs.

REFERENCES

1. *P. Brucker, Yu.N. Sotskov and F. Werner,* "Complexity of shop-scheduling problems with fixed number of jobs: a survey," Mathematical Methods of Operations Research, Vol. 65, No. 3, 461 – 481 (2007).
2. *J. Adams, E. Balas and D. Zawack,* "The shifting bottleneck procedure for jobshop scheduling," Management Science, Vol. 34, No. 3, 391 – 401 (1988).
3. *E. Balas and A. Vazacopoulos,* "Guided local search with shifting bottleneck job shop scheduling," Management Science, Vol. 44, No. 2, 262 – 275 (1998).
4. *V.S. Tanaev, Yu.N. Sotskov and V.A. Strusevich,* "Scheduling Theory: Multi-Stage Systems," Kluwer Academic Publishers, Dordrecht, The Netherlands (1994).