Fast Edge-Orientation Heuristics for Job-Shop Scheduling Problems with Applications to Train Scheduling

Omid Gholami[1], Yuri N. Sotskov[2] and Frank Werner[3*]

[1]Islamic Azad University, Mahmudadad Branch
Moalem, Mahmudabad, Iran
[2]United Institute of Informatics Problems
National Academy of Sciences of Belarus
Belarus, Minsk
[3]Faculty of Mathematics, Otto-von-Guericke-University
PSF 4120, 39106 Magdeburg, Germany
e-mails: gholami@iaumah.ac.ir; sotskov@newman.bas-net.by; frank.werner@ovgu.de

Abstract

A train scheduling problem in a single-track railway can be modeled as a job-shop scheduling problem. We use a mixed graph model for such a job-shop problem with appropriate criteria. There are several performance evaluations for a train schedule. Optimizing a train schedule subtends minimizing total tardiness of the trains, minimizing the sum of train transit times, minimizing the makespan for a train schedule, etc. Since the corresponding job-shop problems with the above three criteria are NP-hard, several constructive heuristics have been developed using different priorities based on the release times of the jobs, the job due-dates and the job completion times. Experiments on a computer were used for evaluating the quality and efficiency of the heuristic algorithms developed for appropriate job-shop problems. The release times, due-dates and completion times of the jobs have been used as input parameters (priorities) to see the effect of them on the quality of the schedules with different objective functions. The efficiency of the developed heuristics was tested on a set of 118 randomly generated instances of small and large sizes with up to 2000 operations.

Keywords: train scheduling, single-track railway, job-shop problem, disjunctive graph, heuristics

1. Introduction

This paper addresses the problem of generating an efficient schedule of passenger and freight trains in a single-track railway. We use the terminology from [11] for train scheduling and that from [17] for machine scheduling.

In the world, the railway traffic is increasing from year to year. The employment of railroads grows both for passenger and freight transportation. When the density of train moving is increasing, the train schedule becomes more difficult both as the generation and control are concerned. During the last decades, a lot of new algorithms and software have been developed and published in the OR literature and in the special literature in order to produce a better tool for generating an accurate and reliable train schedule.

* corresponding author

In this paper, it is shown how one can find a train schedule for a single-track railway which is close to an optimal one for three objective functions. A possible way to achieve a proper train schedule uses job-shop scheduling [4, 12, 16], although job-shop problems are fairly complicated since they belong to the class of NP-hard problems [2, 3, 17]. In order to achieve a practical size of a job-shop problem, which can be solved within a reasonable time, we propose and test several constructive heuristics for three objective functions which are based on the orientation of edges in a mixed graph and which turn out to be appropriate for train scheduling. In Section 2, we give a brief literature review. In Sections 3 - 5, we consider a railway network provided that a pair of sequential stations can be connected by at most one single-track (a railroad section). In particular, this is the case for most railway systems in countries of the Middle East. In Section 6, we present computational results for 118 small and large instances. Section 7 gives some concluding remarks.

2. Literature Review

In [19], a resource-constrained project scheduling was used for a single-track timetabling problem. Both the track segments and stations were modelled as limited resources. A branch-and-bound algorithm has been developed in order to obtain a feasible train timetable with a guaranteed level of optimality. A lower bound based on Lagrangian relaxation was used to relax the segment and station capacity constraints. A lower bound was used to estimate the least train delay. An upper bound was constructed via a beam search heuristic. In [5], a heuristic algorithm was developed for train scheduling in a single-track railway under the assumption that all trains moving in the same direction must have the same speed. A greedy heuristic was proposed based on a local optimality criterion in the event of a potential crossing conflict.

The paper [10] was devoted to train scheduling problems when prioritized trains and non-prioritized trains are simultaneously traversed in a single-track railway. No-wait conditions arise because the prioritized trains (e.g., an express passenger train has a higher priority) should traverse continuously without interruptions. Non-prioritized trains (e.g., a freight train) are allowed to either enter the next section immediately (if it is free) or to remain in a section until the next section on the route becomes available. A generic algorithm has been developed to construct a feasible train timetable in terms of the given train order. The proposed algorithm comprises several recursively used procedures to guarantee the feasibility by satisfying the no-waiting, a deadlock-free condition, and a conflict-free constraint.

Szpigel [16] was the first who identified the similarities between a job shop problem and train scheduling in a single-track railway. The former was solved in [16] using a branch-and-bound algorithm, where the initial linear programming problem excludes the order constraints. Branching is required if the current solution contains trains which are in a conflict (i.e., when trains turn out to be located on the same railroad section at the same time). The objective was to minimize the weighted sum of the train transit times. Computational results for 5 single-track sections and 10 trains have been reported. The same problem was considered in [6] via binary mixed integer programming similarly to

that considered in [8]. The temporal constraints were identical to those used in [16]. The objective was to minimize the deviation from the ideal arrival times and the departure times for the trains to be scheduled. In [13], a job-shop problem was used to solve the train scheduling problem, where a route was interpreted as follows: The route is a sequence of the facilities the train must cross from the origin to the destination. Assuming that the train trips are jobs to be scheduled, which require elements of the infrastructure as restricted resources, it was done by mapping the initial problem into a special case of a job-shop problem. In order to solve the job-shop problem, a constraint programming approach has been developed. A support for finding quickly a good schedule was offered by an original separation and a bound-and-search heuristic. To improve the time performance, a surrogate objective function was used which had a smaller domain than the actual objective function.

In [7], a discrete-event model was used to schedule the traffic on a railway network. This model was computationally efficient and generated near optimal schedules with respect to a number of time-of-travel-related criteria. In [4], train scheduling was interpreted in terms of a job-shop problem with parallel machines. A disjunctive graph model was used in several algorithms with the makespan objective. It was demonstrated that solutions with a good quality may be obtained within a reasonable CPU-time.

3. Problem settings and testing

One of the main problems in the management of a railway network is the train scheduling (timetabling) problem, in which it is necessary to determine a schedule (timetable) for a set of given trains that does not violate the railway constraints. This problem has to be solved at the tactical level of the railway planning process [11]. For the case of a single-track railway, train scheduling may be interpreted as the following job-shop problems.

There are n jobs $J = \{J_1, J_2, \dots, J_n\}$ to be processed on m different machines $M = \{M_1, M_2, \dots, M_m\}$. The time $p_{ij} > 0$ needed for processing an operation O_{ij} of a job $J_i \in \mathcal{J}$ on the corresponding machine $M_v \in \mathcal{M}$ is known. Operation preemptions are not allowed, and the machine routes $\mathcal{O}^i = (O_{i1}, O_{i2}, \dots, O_{in_i})$ for the jobs $J_i \in \mathcal{J}$ may be different. A job $J_i \in \mathcal{J}$ is available for processing from time-point $r_i \geq 0$. The time-point d_i defines a due-date for completing the job J_i. A machine $M_k \in \mathcal{M}$ can process a job $J_i \in \mathcal{J}$ at most once. So, 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 of the set $\mathcal{M}$, i.e., inequality $n_i \leq m$ holds (such a problem is called a classical job-shop).

One objective is to find a schedule minimizing the sum $\gamma = \sum_{i=1}^{n} T_i$ of the tardiness times $T_i = \max\{0, C_i - d_i\}$ of 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 job-shop problem is denoted as $J|r_i|\sum T_i$. If $\gamma = \sum_{i=1}^{n} C_i$, then this problem is denoted as $J|r_i|\sum C_i$. If $\gamma = \max^{n}{}_{i=1} C_i$, then it is denoted as $J|r_i|C_{max}$.

The problems $J|r_i|\sum T_i$, $J|r_i|\sum C_i$ and $J|r_i|C_{max}$ arise in train scheduling for a single-rack railway: To determine the best train schedule among those which do not violate the single-track capacities. For passenger trains, the criteria $\sum T_i$ and $\sum C_i$ are more important than C_{max} while for freight trains, the criteria C_{max} and $\sum C_i$ are more important than $\sum T_i$.
In a job-shop approach to train scheduling, the trains and railroad sections are synonymous with the jobs $J_i \in \mathcal{J}$ and the machines $M_j \in \mathcal{M}$, respectively. An operation O_{ij} is regarded as a movement of a train $J_i \in \mathcal{J}$ across a railroad section $M_v \in \mathcal{M}$ (in the route $\mathcal{O}^i$, machine M_v processes operation O_{ij}). The positive number p_{ij} denotes the time required for the train $J_i \in \mathcal{J}$ to travel through the railroad section $M_v \in \mathcal{M}$. The non-negative number r_i denotes the departure time of the train $J_i \in \mathcal{J}$, which is given in the official train timetable. The positive number d_i denotes the official arrival time of the train $J_i \in \mathcal{J}$ (a due-date for the desired completion time C_i of a job $J_i \in \mathcal{J}$) at the terminal station in the route $\mathcal{O}^i$. Note that for train scheduling, usually the inequality $m \geq n$ holds.

The problems $J|r_i|\sum T_i$, $J|r_i|\sum C_i$ and $J|r_i|C_{max}$ are complicated in the computational sense since their special cases belong to the class of NP-hard problems [17]. In order to achieve a practical size of a classical job-shop problem, which can be solved heuristically within a reasonable time, we have coded a shifting bottleneck algorithm, which was originated in [1] for the job-shop problem $J||C_{max}$. However, testing the program realizing a shifting bottleneck algorithm for the problem $J|r_i|\sum T_i$ showed an unsatisfactorily large CPU-time when the number n of trains was large and the number m of railroad sections was no less than n [14]. Computational experiments showed that this algorithm can handle 125 operations (e.g., 25 trains on 5 railroad sections or 5 trains on 25 railroad sections) within half an hour of CPU time. For larger job-shops with $m \geq n$ (what is typical for train scheduling problems), the CPU-time grows quickly. We also observed that meta-heuristic algorithms are often time-consuming (e.g., a survey of genetic algorithms for shop scheduling algorithms can be found in [18]).

In Sections 4 and 5, we develop heuristic edge-orientation algorithms, which run faster than the shifting bottleneck algorithm providing a quality of the objective function values which is close to the quality of the schedules constructed by the shifting bottleneck algorithm.

4. Mixed graph model

The problems $J|r_i|\sum T_i$, $J|r_i|\sum C_i$ and $J|r_i|C_{max}$ described in Section 3 can be formulated using a mixed graph model $G = (\mathcal{Q}, \mathcal{C}, \mathcal{D})$ [17] or a disjunctive graph model [15].

Let $\mathcal{Q}$ denote the set of operations O_{ij}, $J_i \in \mathcal{J}$, $j \in \{1, 2, \dots, n_i\}$, to be executed by the machines $\mathcal{M}$ and a dummy operation O_{00} associated with the beginning of a schedule and n dummy operations O_{i,n_i+1} associated with the completion of the jobs $J_i \in \mathcal{J}$.

Two operations O_{ij} and O_{lk}, which have to be executed by the same machine $M_u \in \mathcal{M}$, cannot be simultaneously processed by this machine. This restriction is presented by an edge $[O_{ij}, O_{lk}] \in \mathcal{D}$. Two consecutive operations O_{ij} and $O_{i,j+1}$ of the same job $J_i \in \mathcal{J}$ are connected by an arc $(O_{ij}, O_{i,j+1}) \in \mathcal{C}$, where $1 \leq j \leq n_i - 1$. The arc $(O_{ij}, O_{i,j+1})$ means that operation $O_{i,j+1}$ has to be started after the completion of operation O_{ij}. The processing time p_{ij} is prescribed to the arc $(O_{ij}, O_{i,j+1}) \in \mathcal{C}$, and the two processing times p_{ij} and p_{lk} are prescribed to the edge $[O_{ij}, O_{lk}] \in \mathcal{D}$. For the dummy operation $O_{00} \in \mathcal{Q}$, the arc (O_{00}, O_{i1}) with the weight r_i is included into the set $\mathcal{C}$ for each job $J_i \in \mathcal{J}$. For the dummy operation $O_{i,n_i+1} \in \mathcal{Q}$, the arc (O_{in_i}, O_{i,n_i+1}) with the weight p_{in_i} is included into the set $\mathcal{C}$.

The problems $J|r_i|\sum T_i$, $J|r_i|\sum C_i$ and $J|r_i|C_{max}$ are modelled by a mixed graph $G = (\mathcal{Q}, \mathcal{C}, \mathcal{D})$. The due-dates d_i are used when calculating the objective function $\gamma = \sum_{i=1}^{n} T_i$ for a schedule constructed.

Since operation preemptions are not allowed, a schedule on a mixed graph $G = (\mathcal{Q}, \mathcal{C}, \mathcal{D})$ may be defined as a sequence of the starting times $s_t = (s_{00} = 0, {s_1}^1, {s_1}^2, \ldots, {s_1}^{n_1}, {s_1}^{n_1+1}, \ldots, {s_n}^1, {s_n}^2, \ldots, {s_n}^{n_n}, {s_n}^{n_n+1})$ of all the operations $\mathcal{Q}$ such that the conjunctive constraint

$${s_l}^k - {s_i}^j \geq p_{ij} \tag{1}$$

has to be satisfied for each arc $(O_{ij}, O_{lk}) \in \mathcal{C}$, and the disjunctive constraint

$$\text{either } {s_l}^k - {s_i}^j \geq p_{ij} \text{ or } {s_i}^j - {s_l}^k \geq p_{lk} \tag{2}$$

has to be satisfied for each edge $[O_{ij}, O_{lk}] \in \mathcal{D}$.

Using the above weighted mixed graph $G = (\mathcal{Q}, \mathcal{C}, \mathcal{D})$, to define a feasible sequence s_t. of the starting times, one has to replace each edge $[O_{ij}, O_{lk}] \in \mathcal{D}$ either by the arc (O_{ij}, O_{lk}) with the weight p_{ij} or by the arc (O_{lk}, O_{ij}) with the weight p_{lk} respecting the disjunctive constraint (2) in such a way that no circuit arises in the obtained digraph. As a result, the set of edges $\mathcal{D}$ will be substituted by the chosen set $\mathcal{D}_t$, the mixed graph $G = (\mathcal{Q}, \mathcal{C}, \mathcal{D})$ will be transformed into a circuit-free digraph $G_t = (\mathcal{Q}, \mathcal{C} \cup \mathcal{D}_t, \emptyset)$, and an operation sequence for each machine of the set $\mathcal{M}$ will be determined. Since the cardinality of the set $\mathcal{Q}$ is equal to $|Q| = 1 + \sum_{i+1}^{n}(n_i + 1)$, using the critical path method, one can build a unique semiactive schedule defined by the weighted digraph G_t in $O(n^2)$ time. A schedule is called semiactive if no operation O_{ij}, $J_i \in \mathcal{J}$, $j \in \{1, 2, \ldots, n_i\}$, can start earlier without delaying the processing of some operation from the set Q or (and) without altering the processing sequence of the operations on any of the machines $\mathcal{M}$.

The main complexity of the problem $J|r_i|\gamma$ with a regular criterion γ is to find an optimal circuit-free digraph $G_t = (\mathcal{Q}, \mathcal{C} \cup \mathcal{D}_t, \emptyset)$ generated by the mixed graph $G = (\mathcal{Q}, \mathcal{C}, \mathcal{D})$. In other words, it is necessary to find a set of arcs $\mathcal{D}_t$ for substituting the set of edges $\mathcal{D}$ in the mixed graph G such that the objective function γ has the minimal value among all other circuit-free digraphs generated by the mixed graph G via replacing each edge $[O_{ij}, O_{lk}] \in \mathcal{D}$ either by the arc $(O_{ij}, O_{lk}) \in \mathcal{D}_t$ or by the arc $(O_{lk}, O_{ij}) \in \mathcal{D}_t$.

5. Heuristic edge-orientation algorithms

We developed three types of constructive edge-orientation algorithms denoted as Ordinal, Max-PT and Min-PT. The release times, completion times and due-dates are used as priorities in ordering the jobs $J_i \in \mathcal{J}$ for processing on the same machine from the set $\mathcal{M}$. Nine heuristic algorithms of three types with three priority rules are developed.

5.1 Ordinal-algorithm

The Ordinal-algorithm generates a sequence of the operations O_{ij} on different machines of the set $\mathcal{M}$ in the order as they are requested for processing the jobs $J_i \in \mathcal{J}$. In the first iteration, the Ordinal-algorithm finds the first request (i.e., operation O_{i1}) of a job $J_i \in \mathcal{J}$ for the machine $M_u \in \mathcal{M}$ processing operation O_{i1}. Then, depending on which priority rule is used, the Ordinal-algorithm computes either the release time or the completion time or the due-date as the priority of operation O_{i1}. For example, let the Ordinal-algorithm use the release time of operation O_{i1} as its priority. Then the algorithm compares the release time r_{i1} of operation O_{i1} with the release times of all operations O_{jk} of the other jobs $J_i \in \mathcal{J}$, $i \neq j$, on the same machine $M_u \in \mathcal{M}$ processing operation O_{jk}. If the release time r_{i1} is smaller than the release time of the operations of the other jobs on the same machine $M_u \in \mathcal{M}$, then an arc starting from operation O_{i1} and ending in operation O_{jk} has to be added to the digraph $(\mathcal{Q}, \mathcal{C}, \emptyset)$. Otherwise, the symmetric arc (O_{jk}, O_{i1}) has to be added to the digraph $(\mathcal{Q}, \mathcal{C}, \emptyset)$.

The release time r_{uv} denotes the earliest start time of operation O_{uv} which can be computed due to the recursion $r_{uv} = max\{r_{ij} + p_{ij}\}$, where the maximum is taken over all operations $O_{ij} \in \mathcal{Q}$ preceding operation O_{uv} in the digraph already constructed. The release time of the source operation O_{00} is equal to zero.

The above procedure is repeated for the second job request (iteration 2), then for the third job request (iteration 3) and so on until all machine requests have been satisfied. We called this version of the algorithm as Ordinal-SRT (from Shortest Release Time).

The other two versions of the Ordinal-algorithm based on either the completion time priority or the due-date priority are called Ordinal-SCT (Shortest Completion Time) and Ordinal-SDD (Shortest Due-Date), respectively.

5.2 MaxPT-algorithm

The MaxPT-algorithm (Maximum Processing Time) tends to schedule first the jobs that need more processing time on all machines $M_u \in \mathcal{M}$.

In the first step, the MaxPT-algorithm calculates the sum of the processing times (total processing time) of all operations O_{ij} , $j \in \{1, 2, .. n_i\}$ for each job $J_i \in \mathcal{J}$. Before scheduling, the maximum sum of the processing times of a job $J_i \in \mathcal{J}$ is equal to the length of a critical path in the digraph $(\mathcal{Q}, \mathcal{C}, \emptyset)$.
The MaxPT-algorithm sorts the jobs $\mathcal{J}$ in non-increasing order of their total processing times and selects a job with the maximum total processing time to be processed next. The MaxPT-algorithm starts to process the first request (operation O_{i1}) of the job J_i with maximum total processing time, then the second request of the same job and so on until the last request of job J_i. At each operation, the MaxPT-algorithm computes one of the three priorities (either the release time, the completion time or the due-date) depending on the version of the algorithm. The chosen priority is compared with those of all operations of the other jobs on the same machine M_u. Then either the arc (O_{i1}, O_{jk}) or the arc (O_{jk}, O_{i1}) is added to the digraph $(\mathcal{Q}, \mathcal{C}, \emptyset)$ depending on the larger priority of job J_i or job J_j . The added arc defines the order of processing the jobs J_i and J_j on machine M_u.

Then the MaxPT-algorithm repeats the same procedure for the other jobs that are sorted by non-increasing sums of their processing times. We call this version of the algorithm which uses the release time as priority as MaxPTRT-algorithm (Maximum Processing Time, Release Time).

The MaxPTCT-algorithm (Maximum Processing Time and Completion Time) is another version that compares the job completion times as priorities, and the MaxPTDD-algorithm (Maximum Processing Time, Due-Date) compares the due-dates as priorities.

5.3 MinPT-algorithm

The MinPT-algorithm (Minimum Processing Time) is basically similar to its counterpart, the MaxPT-algorithm but in contrast to the latter one, the MinPT-algorithm schedules first the job $J_i \in \mathcal{J}$ that needs the smallest total processing time on all machines $\mathcal{M}$.

The MinPT-algorithm sorts the jobs in non-decreasing order of their total processing times and then schedules the jobs on each machine $M_u \in \mathcal{M}$ in non-decreasing order of the corresponding priorities.

The three versions of the MinPT-algorithm are the MinPTRT-algorithm (Minimum Processing Time, Release Time), the MinPTCT-algorithm (Minimum Processing Time, Completion Time) and the MinPTDD-algorithm (Minimum Processing Time, Due-Date).

6. Computational results

Three versions of the three types of algorithms (Ordinal, Max-PT and Min-PT) have been coded in Borland Delphi. For the computational experiments, a laptop computer with the following specification has been used: Intel, coreTM 2 Duo, CPU T6400, 2.00 GHz and 2GB Internal Memory, Windows 7, Ultimate 32 bit. We were mainly interested in investigating experimentally the effect of choosing different types of algorithms and different priorities for the three objective functions and therefore, we compare the nine algorithms relative to each other.

6.1 Small instances

First, we considered 80 small instances with at most 144 operations per instance. In particular, we randomly generated job-shop problems of size $n \times m$, where $n = m \in \{5, 6, \dots, 12\}$ to see the effect of these algorithms on different objective functions for job-shop problems. The computational times for each of these small instances is less than 1 second.

First, we compared the makespan objective function for 80 randomly generated instances of the problem $J|r_i|C_{max}$ with $n = m$. We compared the makespan values obtained by the nine algorithms developed for instances with the same input data. Each series includes 10 randomly generated instances. The results are presented in Fig. 1, which shows that the quality of a schedule obtained by the algorithms generally depends on the input data, but both the OrdinalSCT-algorithm and the OrdinalSRT-algorithm can be recommended for the makespan criterion.

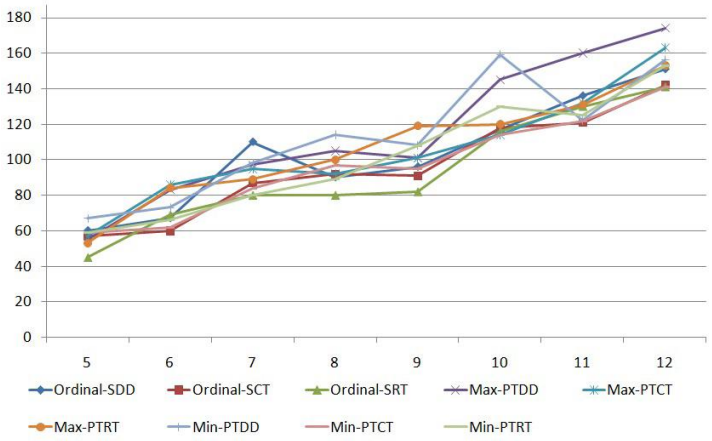

Figure 1: Objective function values of the obtained schedules for job-shop problems with the criterion C_{max}.

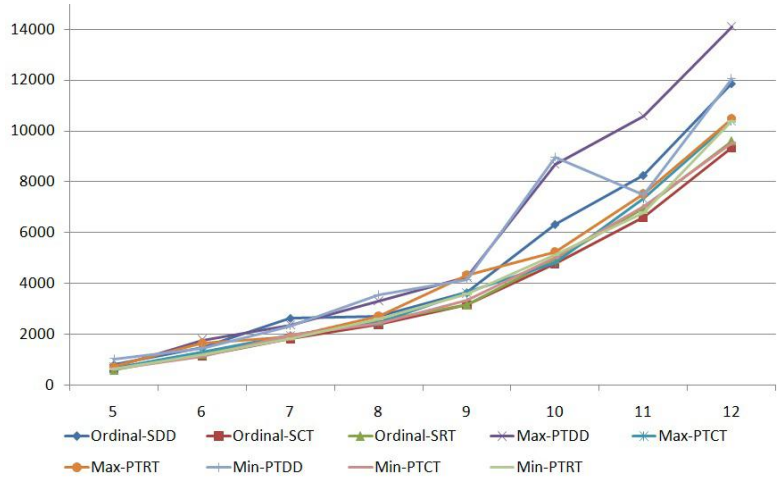

Figure 2: Objective function values of the obtained schedules for job-shop problems with the criterion $\sum C_i$.

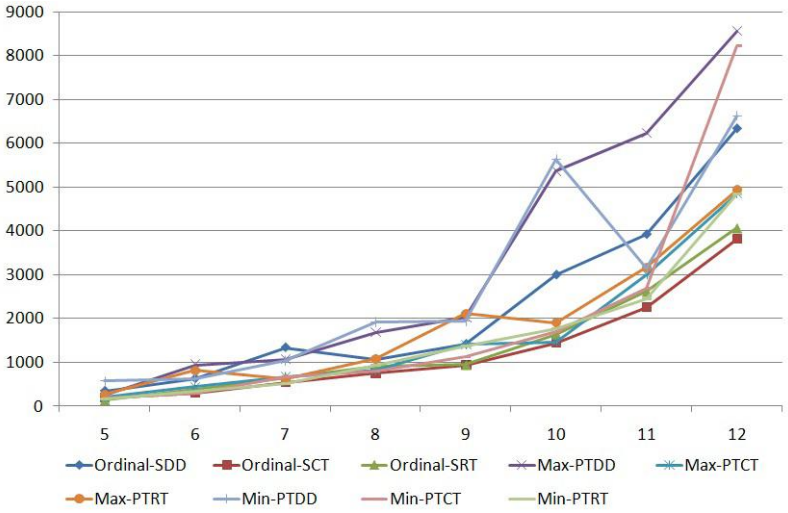

Figure 3: Objective function values of the obtained schedules for job-shop problems with the criterion $\sum T_i$.

Figure 2 gives the function values for the nine algorithms with the objective function $\sum C_i$. It can be seen that the OrdinalSCT-algorithm provides schedules with the best quality among the algorithms tested. In addition, the OrdinalSRT-algorithm can also be recommended.

Then we evaluated the objective function $\sum T_i$. Figure 3 shows that the OrdinalSCT-algorithm again provides schedules with the best quality among the algorithms tested.

6.2 Medium and large instances

Next, we considered 38 medium and large instances with up to 2000 operations per instance to compare the nine edge-orientation heuristics. First, we considered 10 instances with $n = 20$ jobs. For each value of $m \in \{10, 20, \ldots, 100\}$ we randomly generated one instance. So all but one of these instances satisfy the inequality $n \leq m$ which is typical for train scheduling problems. In Table 1, we compare the ranks of the nine edge-orientation algorithms (i.e., the best algorithm gets rank 1, the second best algorithm gets rank 2, and so on, while the worst algorithm gets rank 9). For each objective function, we give the average ranks (ave) and the numbers of best function values (nbv) obtained by the particular algorithms among the 10 instances considered.

It can be observed that mostly the DD variants (with the due-dates as priorities) work poor. On the other side, particularly for the sum criteria, the CT variants work well in general. However, the Ordinal-algorithm is not superior for the makespan criterion and even for the sum criteria, the Ordinal SRT-algorithm is not as good as observed for the small instances while the Ordinal SCT-algorithm works well for these instances.

Next, we generated 8 instances with a fixed number $m = 20$ of machines. For $n \in \{20, 30, 40, 50, 60, 70, 80\}$, we randomly generated one instance. Here, all but one instance satisfy the inequality $n \geq m$. For these instances, again the average ranks (ave) and the numbers of best values (nbv) obtained by the particular algorithms are given in Table 2.

Table 1: Results for randomly generated instances with $n = 20$ jobs and $m \in \{10, 20, \ldots, 100\}$ machines

n = 20 Algorithm	**C_{max} ave**	**nbv**	**ΣC_i ave**	**nbv**	**ΣT_i ave**	**Nbv**
OrdinalSDD	6.1	0	8.4	0	8.4	0
OrdinalSCT	3.95	3	2.5	3	2.4	3
OrdinalSRT	5.2	1	3.9	1	4.0	1
MaxPTDD	5.45	0	8.0	0	8.0	0
MaxPTCT	6.75	0	3.4	1	3.4	1
MaxPTRT	6.05	1	4.0	1	4.0	1
MinPTDD	4.85	0	7.6	0	7.6	0
MinPTCT	2.75	2	3.5	2	3.5	2
MinPTRT	3.9	3	3.7	2	3.7	2

Table 2: Results for randomly generated instances with
$n \in \{10, 20, \ldots, 80\}$ jobs and $m = 20$ machines

m = 20 Algorithm	**C_{max} ave**	**nbv**	**ΣC_i ave**	**nbv**	**ΣT_i ave**	**Nbv**
OrdinalSDD	7.125	0	8.5	0	8.5	0
OrdinalSCT	4.25	1	4.125	0	4.125	0
OrdinalSRT	5.375	1	3.0	2	3.0	2
MaxPTDD	6.125	0	8.125	0	8.0	0
MaxPTCT	5.125	0	3.125	1	3.25	1
MaxPTRT	3.875	1	2.375	3	2.25	3
MinPTDD	4.875	1	7.375	0	7.5	0
MinPTCT	3.625	1	4.0	1	4.0	1
MinPTRT	4.625	3	4.375	1	4.375	1

From Table 2, it can be observed that the DD variants work again poor, particularly for the sum criteria. In contrast to the instances evaluated in Table 1, now the SRT variants work better than the SCT variants for the sum criteria, where the OrdinalSCT- and the MaxPTCT-algorithms can be recommended. It can also be seen that the situation is a bit different for the makespan criterion, where e.g. the OrdinalSRT-algorithm is not so good).

Table 3: Average CPU-times in seconds for instances with n jobs and m machines

m =	10	20	30	40	50	60	70	80	90	100
n = 20	1	2	5	7	12	20	28	32	45	60
n =	10	20	30	40	50	60	70	80		
m = 20	1	2	10	27	81	160	360	1080		

Table 4: Results for the Lawrence instances La01 - La20 for the criterion C_{max}

Algorithm	**ave**	**nbv**	**brank**	**wrank**
OrdinalSDD	7.3	0	3	9
OrdinalSCT	4.6	2	1	9
OrdinalSRT	3.15	6	1	8
MaxPTDD	6.6	0	3	9
MaxPTCT	5.2	0	3	9
MaxPTRT	4.65	4	1	9
MinPTDD	6.55	1	1	9
MinPTCT	4.3	3	1	9
MinPTRT	2.65	4	1	6

In Table 3, we summarize the average computational times of the nine algorithms in dependence on the number n of jobs and the number m of machines. It can be seen that these times only moderately increase with m, if the number of jobs is constant. In this case, even instances with 100 machines can be solved within a minute. On the other side, if m is constant, these times grow faster with the number of jobs. The reason for this behavior is that for large n, the numbers of edges to be oriented is much larger than in the case of a large value of m.

Although the heuristics presented in this paper were mainly developed for train scheduling problems (where typically $n \leq m$ holds and often sum criteria are considered), we also compared the nine algorithms on the 20 benchmark instances La01 - La20 given by Lawrence [9]. The results are given in Table 4, where the average ranks of the algorithms (ave), the numbers of best values obtained by the particular variants (nbv) together with the best rank (brank) and the worst rank (wrank) are given. It can also be seen that there is no clear superiority of particular variants. Although mostly the quality of the more time consuming shifting bottleneck procedure was not reached, nevertheless there exist also makespan instances, where a good edge orientation algorithm reaches the quality of the shifting bottleneck procedure or even a better objective function value.

Table 5: Comparison of the procedures for medium and large instances

Algorithm	**C_{max} m=20**	**ΣC_i m=20**	**ΣT_i m=20**	**C_{max} n=20**	**ΣC_i n=20**	**ΣT_i n=20**	**Lawrence**
OrdinalSDD	9	9	9	9	9	9	9
OrdinalSCT	3	5	5	3	1	1	4
OrdinalSRT	7	2	2	5	5	5.5	2
MaxPTDD	8	8	8	6	8	8	8
MaxPTCT	6	3	3	9	2	2	6
MaxPTRT	2	1	1	7	6	5.5	5
MinPTDD	5	7	7	4	7	7	7
MinPTCT	1	4	4	1	3	3	3
MinPTRT	4	6	6	2	4	4	1

Finally, we give an overall evaluation of the nine algorithms for the medium and large instances. In Table 5, we give the average ranks of each of the algorithms for the seven different types of problems (three criteria for fixed n, three criteria for fixed m and the Lawrence instances La01 - La20 for the criterion C_{max}). From Table 5, we see again that all DD (due-date) variants work poor. For the instances with fixed n (where mostly $n \leq m$ holds), the CT variants work well in general. In Table 5, we tried to give a pattern (drawn in bold face) of variants of edge-orientation algorithms which can be recommended for the particular types of problems. In particular, we can recommend both Ordinal-SCT and MaxPTCT for instances with $n \leq m$. In contrast, for the larger makespan instances, the variants MinPTCT and MinPTRT seems to work good.

7. Concluding Remarks

The problem of finding an optimal train schedule includes several criteria. We considered three of them and developed nine heuristic edge-orientation algorithms to solve the corresponding job-shop problems. In particular, we coded different variants of heuristic algorithms and compared three parameters for job-shop scheduling to find suitable heuristic algorithms for three objective functions.

The computational results showed that the selection of an appropriate algorithm depends on the type of problem (ratio of n and m, objective function, size of the instance). However, for the case $n \leq m$, which is typical for train scheduling problems in a single-track railway, often the OrdinalSCT-algorithm generates a good schedule when minimizing the sum of job completion times or the sum of job tardiness. Since the edge-orientation heuristics are rather fast, one can apply several good variants to an instance. We also observed in our tests that the use of a more complicated algorithm (like the shifting bottleneck one) for solving train scheduling problems needed more CPU-time for the case when $m \geq n$ (the bottleneck machine is often shifted) with only a slight improvement of the objective function values.

For future research, we recommend to compare more parameters for those objective functions appropriate for train scheduling. Note that the OrdinalSCT and OrdinalSRT-algorithms can be generalized to the weighted objective functions $\sum w_i T_i$ and $\sum w_i C_i$ allowing a scheduler to take into account different priorities of the trains. In addition, the inclusion of a learning stage in the edge-orientation procedure seems promising.

References

[1] Adams, J., Balas, E. and Zawack, D., 1988, The shifting bottleneck procedure for job-shop scheduling. *Management Science*, **34**, 391 - 401.

[2] Brucker, P., Kravchenko, S. and Sotskov, Y., 1997, On the complexity of two-machine job-shop scheduling with regular objective functions. *Operations Research Spektrum*, **19**, 5 - 10.

[3] Brucker, P., Sotskov, Y. and Werner, F., 2007, Complexity of shop scheduling problems with fixed number of jobs: a survey, *Mathematical Methods of Operations Research*, **65**, 461 - 481.

[4] Burdett, B. and Kozan, E., 2010, A disjunctive graph model and framework for constructing new train schedules, *European Journal of Operational Research*, **200**, 85 - 98.

[5] Cai, X. and Goh, C., 1994, A fast heuristic for the train scheduling problem, *Computers & Operations Research*, **21**, 499 - 510.

[6] Carey, M. and Lookwood, D., 1995, A model, algorithms and strategy for train pathing, *Journal of the Operations Research Society*, **46**, 988 -1005.

[7] Dorfman, M. and Medanic, J., 2004, Scheduling trains on a railway network using a discrete event model of railway traffic, *Transportation Research Part B*, **38**, 81 - 98.

[8] Jovanovic, D. and Harker, P., 1991, Tactical scheduling of rail operations: the scan i system, *Transportation Science*, **25**, 46 - 64.

[9] Lawrence, S., 1984, *Supplement to resource constrained scheduling: An experimental investigation of heuristic scheduling techniques*, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburg.

[10] Liu, S. and Kozan, E., 2011, Scheduling trains with priorities: a no-wait blocking parallel-machine job-shop scheduling model, *Transportation Science*, **45**, 175 - 198.

[11] Lusby, R., Larsen, J., Ehrgott, M. and Ryan, D., 2011, Railway track allocation: models and methods, *Operations Research Spectrum*, **33**, 843- 883.

[12] Mascis, A. and Pacciarelli, D., 2002, Job shop scheduling with blocking and no-wait constraints, *European Journal of Operational Research*, **143**, 498 - 517.

[13] Mladenovic, S. and Cangolovic, M., 2007, Heuristic approach to train rescheduling, *Yugoslav Journal of Operations Research*, **17**, 9 - 29.

[14] Sotskov, Y. and Gholami, O., 2012, Shifting bottleneck algorithm for train scheduling in a single-track railway, *Proccedings of the 14th IFAC Symposium on Information Control Problems, Part 1, Bucharest / Romania*, 87 - 92.

[15] Sussmann, B., 1972, Scheduling problems with interval disjunctions, *Mathematical Methods of Operations Research*, **16**, 165 - 178.

[16] Szpigel, B., 1973, Optimal train scheduling on a single line railway, *Operations Research*, **72**, 344 - 351.

[17] Tanaev, V., Sotskov, Y. and Strusevich, V., 1994, *Scheduling Theory: Multi-Stage Systems*, Kluwer Academic Publishers, Dordrecht, The Netherlands.

[18] Werner, F., 2011, *Genetic algorithms for shop scheduling problems: a survey*, Preprint 31/11, Faculty of Mathematics, Otto-von-Guericke-University Magdeburg, 66 p.

[19] Zhou, X. and Zhong, M., 2007, Single-track train timetabling with guaranteed optimality: branch-and-bound algorithms with enhanced lower bounds, *Transportation Research Part B*, **21**, 320 - 341.