Scheduling trains as a blocking parallel-machine job shop scheduling problem

Shi-Qiang Liu and Erhan Kozan*
School of Mathematical Sciences, Queensland University of Technology
2 George St GPO Box 2434, Brisbane Qld 4001 Australia

Abstract

In this paper, the train scheduling problem is modelled as a *Blocking Parallel-Machine Job-Shop-Scheduling* (BPMJSS) problem. In the model, trains, single-track sections and multiple-track sections respectively are synonymous with jobs, single machines and parallel machines; and an operation is regarded as the movement/traversal of a train across a section. Due to the lack of buffer space, the real-life case should consider blocking or hold-while-wait constraints, which means that a track section cannot release and must hold the train until next section on the routing becomes available. Based on literature review and our analysis, it is very hard to find a feasible complete schedule directly for BPMJSS problems. Firstly, a *parallel-machine job-shop-scheduling* (PMJSS) problem is solved by an improved *Shifting Bottleneck Procedure* (SBP) algorithm without considering blocking conditions. Inspired by the proposed SBP algorithm, *Feasibility Satisfaction Procedure* (FSP) algorithm is developed to solve and analyse the BPMJSS problem, by an alternative graph model that is an extension of the classical disjunctive graph models. The proposed algorithms have been implemented and validated using real-world data from Queensland Rail. Sensitivity analysis has been applied by considering train length, upgrading track sections and increasing train speed. The outcomes show that the proposed methodology would be a very useful tool for the real-life train scheduling problems.

Keywords: blocking; parallel machine; job shop scheduling; train scheduling

1. Introduction

The railway constitutes an important mode of transportation for both freight and passengers in many countries. The railway industry is a capital intensive industry with large investment in equipment and employees. In addition, operating a railway is a very complex decision-making process due to the need to schedule several hundred trains over thousands of kilometres distances. However, even a small percentage of improvement in the efficiency of the overall operation may bring significant financial return.

Furthermore, Australia is the world's largest coal exporting country. Many large coal mining operations in Queensland mainly rely on rail network to transport coal from various mines to coal terminals at ports for shipment. Over the last few years, due to the fast growing demand, the coal rail network is becoming one of the worst bottlenecks in rail industry. In this context, the railway industry in Australia demands more new features in the planning and scheduling process and is keen to implement better modelling and solution techniques.

* Corresponding author. Tel: +61 7 3138 1029; Fax: +61 7 3138 2310.
Email addresses: e.kozan@qut.edu.au (E. Kozan).

Therefore, the current situation provides great incentives for pursuing better optimisation and control strategies for the operation of the rail transportation system. By generating better rail schedules, it is possible to increase the utilisation rate of the rail network and reduce the transportation cost.

2. Previous Research

This paper aims to achieve a significant efficiency improvement in rail network on the basis of the development of modelling approaches, algorithms analysis, and solution techniques. Some previous research in this field is summarised below.

Higgins, Ferreira, and Kozan [1] modelled a single line train operations when the priority of each train in a conflict depends on an estimate of remaining crossing and overtaking delay. This priority is used in a branch-and-bound algorithm to allow the resolution of conflicts quickly. Considering that investment decision on upgrading the number and location of sidings have a significant impact on both rail reliability and rail profitability, they put forward a model to determine the optimal position of a set of sidings on a single track rail corridor. The sidings are positioned to minimise the total delay and train operating costs of a given cyclic train schedule, with the allowance of non-constant train velocities and non-uniform departure times. Abdekhodaee et al. [2] investigated the integration of scheduling a rail network with some of operations in a coal terminal system with limited capacity, because the rail network and terminal systems are tightly-coupled and experience a high service demand for the expensive infrastructure which makes efficient operations essential. They discussed the merits and disadvantages of devising such integration and the emphasis on promoting coordination between the operational functions of these two systems. Kozan and Burdett [3] proposed approaches to the determination of railway capacity and the significance of some factors on capacity. An accurate model is developed to calculated railway capacity considering previously unaddressed aspects for capacity determination. Capacity and pricing are two key issues for organizations involved with open track access regimes. A train access charging methodology is therefore developed and incorporated into the railway capacity determination model. Burdett and Kozan [4] developed capacity analysis techniques and methodologies for estimating the absolute traffic carrying ability for railway system under a wide range of defined operational conditions, which include the proportional mix of trains, the directions, the length of trains, the planned dwell times of trains, the presence of crossing loops, and intermediate signals in corridors.

Recently, the public railway sector in many parts of the world has increased the awareness of the need for quality service that must be offered to its customers. In this context, Ingolotti et al. [5] developed a software system, named as Decision Support System (DSS), for plotting and solving the single-track railway scheduling problem (STRSP) efficiently. The STRSP problem is formulated as a constraint satisfaction problem (CSP), which is solved by using different stages to translate problems into mathematical models by means of mixed integer programming tools. The DSS allows the users (Railway companies) to interactively specify the parameters of the STRSP problem and guarantees that the constraints are satisfied and the optimised timetables are obtained. Epstein et al. [6] developed a mathematical programming model to determine the optimal dispatching times for complex rail networks in densely populated metropolitan areas, in which some portions of the rail network may consist of single-track line while other locations may consist of double-track or triple-track lines. The model is solved by a branch-and-bound algorithm with the help of transferring trackage to general network graph and applying propagation rules. They also demonstrated the efficiency of the proposed branch-and-bound algorithm by comparing it to CPLEX, a commercially available integer program solver, on an actual rail network in Los Angeles. Linder and Zimmermann [7] considered minimising the operational cost of train schedules which depend on choosing different train types of diverse speed and cost. A mixed integer programming model was proposed for modelling this train scheduling problem. Although it seems to be

impossible to directly solve the model of practical sizes within a reasonable amount of time, suitable decomposition can be applied to achieve good performance. In the first part of the decomposition, only the train type related constraints stay active. In the second part, the remaining constraints are satisfied using relaxation technique. This decomposition idea provides a cornerstone for an algorithm integrating cutting plane and branch-and-bound to optimise the railway networks in Germany and the Netherlands. Zhou and Zhong [8] dealt with a double-track train scheduling problem with multiple objectives. Focusing on a high-speed passenger rail line in an existing network, the problem is to minimise both, 1) the expected waiting times for high-speed trains (efficiency criterion); and 2) the total travel times of high-speed and medium-speed trains (effectiveness criterion). By applying two practical priority rules to model acceleration and deceleration times, the problem is formulated as a multi-mode flow-shop scheduling problem. A branch-and-bound algorithm with effective dominance rule is developed for the bicriteria scheduling problem, and a beam search algorithm with utility evaluation rules is used to construct non-dominated solutions. The authors illustrated the methodology and evaluated the performances of the proposed algorithm by a case study based on Beijing-Shanghai high-speed railway in China.

Even though recent progress in branch-and-bound (one typical exact algorithm) has lead to exact solution for some combinatorial optimisation problems, most real-world problems are either computationally intractable by their NP-hard nature or sufficiently large so as to preclude the use of exact algorithm. In such cases, many researchers have dealt with train scheduling problems by heuristic algorithms. In recent years, the following papers in the literature have addressed this issue. Higgins, Kozan, and Ferreira [9] initiated applying metaheuristic techniques to solve the single-track railway scheduling problem with respect to the number of conflicts. The heuristics applied include a local search heuristic with an improved neighbourhood structure, genetic algorithms, tabu search, and two hybrid algorithms. Higgins and Kozan [10] presented a model to quantify the expected positive delay for individual passenger trains and track links in an urban rail network. The model specifically addressed direct delay, knock-on delays, and delays at scheduled connections. An iterative refinement algorithm was proposed to find the feasible solution. Model validation was carried out using a real-life suburban train network consisting of 157 trains. Oliveira and Smith [11] modelled the single-track railway scheduling problem as a special case of the job-shop scheduling problem. It was achieved by considering the train trips as jobs, which are scheduled on track sections regarded as machines. A train trip may have many tasks (job operations) that consist of traversing from one point to another on a track. The objective of this model is to minimise the total delay. As the situation of two trains occupying the same section of the track at the same may occur, the conflicts are resolved by applying the shortest processing time (SPT) rule to reschedule the tasks on all track sections on which a conflict is found. Chew et al. [12] developed a computerised train-operator scheduling system based on an optimisation approach, which has been implemented at Singapore Mass Rapid Transit (SMRT). The optimization approach involves a bipartite matching algorithm for the generation of night duties and a tabu search algorithm for the generation of day duties. The system can automate the train-operator scheduling process at SMRT, produce favourable schedules in comparison with the manual process, and handle the multiple objectives inherent in the crew scheduling system. Pacciarelli and Pranzo [13] presented the idea of applying tabu search algorithm to a multiple-track railway scheduling problem by means of the *alternative graph*, which is an extension of the classical disjunctive graph.

3. Blocking Parallel-Machine Job-Shop-Scheduling (BPMJSS)

The railway network, as depicted in Figure 1, consists of a set of single-track sections and a set of multiple-track sections referred to as *Crossing Loops* (*sidings*).

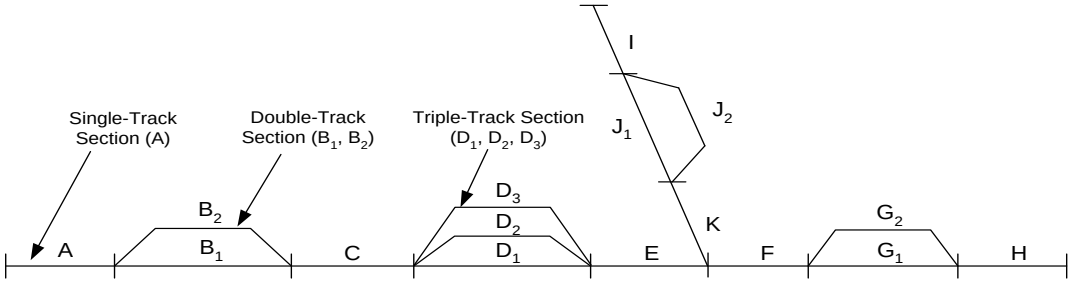

Fig. 1. The Railway Network

The railway network concerned in this research is such that only one train can occupy a single-track section at a time, whereas more than one train can be at a crossing loop (i.e. multiple-track section) at a time as its capacity limit is regarded. Crossing loops are places where trains can stop or slow down in order to let another cross it, or where trains can stop to load or unload cargoes, alight passengers, and manoeuvre crew. Usually, a traversing track section (like A or B_1) is necessarily delimited by at least two signals: one at the beginning and another at the end of section, which will control when a train either can or cannot traverse on that section. This control is to avoid two trains running on the same traversing track section at a time. Due to the lack of buffer or storage space, the real-world case should consider blocking or hold-while-wait constraints, which means that a track section cannot release and must hold the train until next section on the routing becomes available. As a consequence, the train scheduling problem should consider the blocking conditions in process.

According to the above analysis, this train scheduling problem can be modelled as a *Blocking Parallel-Machine Job-Shop Scheduling* (BPMJSS) problem. This is achieved by considering the train trips as jobs, which will be scheduled on single-track sections that are regarded as single machines, and on multiple-track sections that are referred to as parallel machines.

The mathematical programming formulation for BPMJSS is proposed as follows:

Notations

n number of jobs (trains).

m number of machines (sections).

J_i job i ($i = 1, 2, \ldots n$).

M_k machine k ($k = 1, 2, \ldots, m$).

h_k number of units of machine k; default is single machine $h_k = 1$.

u_l the l^{th} unit of machine k ($l = 1, \ldots, h_k$).

o index of sequence position of operation in one job ($o = 1, 2, \ldots, m$).

s_{ilk} starting time of job i on the l^{th} unit of machine k.

p_{ilk} processing time of job i on the l^{th} unit of machine k.

r_{iolk} = 1, if the o^{th} operation of job i requires the lth unit of machine k;
= 0, otherwise.

x_{ilk} = 1, if job i is assigned to the l^{th} unit of machine k;
= 0, otherwise.

y_{ijlk} = 1, if both jobs i and j are assigned to the lth unit of machine k and job i precedes job j (not necessarily immediately);
= 0, otherwise.

w_{ijolk} = 1, if the o^{th} operation of job i requires the l^{th} unit of machine k; and Job j is scheduled on this same unit as its successor;
= 0, otherwise.

$C_{\max}$ maximum completion time or makespan.

L a very large positive number.

The Model

$$\textit{Minimise } C_{\max} \tag{1}$$

The objective function is to minimise the makespan.

Subject to:

$$\sum_{l=1}^{h_k}\sum_{k=1}^{m} r_{iolk}(s_{ilk}+p_{ilk}) \le \sum_{l=1}^{h_k}\sum_{k=1}^{m} r_{i,o+1,l,k}s_{ilk} \quad o=1,2,\ldots,m-1,\forall i. \tag{2}$$

Equation (2) restricts the starting time of $(o+1)^{th}$ operation of job i to be no earlier than its finish time of the o^{th} operation of job i.

$$s_{ilk} \ge s_{jlk} + p_{jlk} + L(y_{ijlk}-1)\ \forall i,j,l,k. \tag{3}$$

Equation (3) restricts that both jobs i and j are processed on the l^{th} unit of machine k and job i precedes job j (not necessarily immediately).

$$s_{jlk} \ge s_{ilk} + p_{ilk} + L(y_{jilk}-1)\ \forall i,j,l,k. \tag{4}$$

Equation (4) restricts that that both jobs i and j are processed on the l^{th} unit of machine k and job j precedes job i (not necessarily immediately).

$$y_{ijlk} + y_{jilk} \le 1\ \forall i,j,l,k. \tag{5}$$

Equation (5) restricts that conditions that job j precedes job i or job i precedes job j at the l^{th} unit of machine k are exclusive.

$$\sum_{l=1}^{h_k}\sum_{k=1}^{m} x_{ilk} = 1 \text{ and } \quad x_{ilk} + x_{jlk} - 1 \le y_{ijlk} + y_{jilk} \quad \forall i,j,l,k. \tag{6}$$

Equation (6) restricts that each unit can process at most one job at a time.

$$\sum_{l=1}^{h_k}\sum_{k=1}^{m} r_{imlk}(s_{ilk}+p_{ilk}) \le C_{\max} \quad \forall i. \tag{7}$$

Equation (7) restricts that the completion time of the m^{th} (i.e. last) operation of each job is no earlier than makespan.

$$s_{ilk}, p_{ilk} \ge 0 \quad \forall i,l,k. \tag{8}$$

Equation (8) satisfies non-negativity condition.

$$\sum_{j=1}^{n}\sum_{l=1}^{h_k}\sum_{k=1}^{m} r_{jolk}s_{jlk}w_{ijolk} \ge \sum_{l=1}^{h_k}\sum_{k=1}^{m} r_{i,o+1,l,k}s_{ilk}\ ,\ i \ne j;\ o=1,2,\ldots,m-1;\forall i. \tag{9}$$

Equation (9) defines the blocking conditions and satisfies that the starting time of the successor on the same machine should be greater and equal to the starting time of the successor of the same job, for each operation.

4. Alternative Graph for BPMJSS

From the point of view of the modelling techniques, most research works in scheduling are based on the disjunctive graph formulation of Roy and Sussman [14]. The disjunctive graph model has been extensively studied in order to develop efficient solution algorithms for solving many scheduling problems. However, a strong limitation that still remains in this classical disjunctive graph is that it disregards the capacity of intermediate buffers between machines. In fact, in many real-life situations especially for train scheduling, the inter-machine buffer capacity has to be taken into account. To incorporate this restriction, the disjunctive graph formulation can be adapted to a more general graph model called *alternative graph* from Masics and Pacciarelli [15].

In the alternative graph, we distinguish two types of operations, namely *ideal* and *blocking*. An ideal operation remains on a machine from its starting time to its completion time, and then at once leaves this machine which becomes immediately available for processing other operations. On the contrary, a blocking operation may remain on a machine even after its completion time, thus blocking it.

Figures 2 and 3 illustrate how a disjunctive graph transforms to the alternative graph for a three-job (or three-train) four-machine (or four-section) blocking flow-shop scheduling instance.

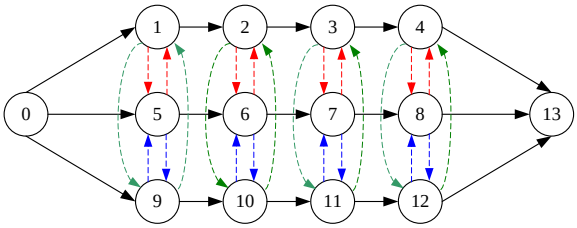

Fig.2. The disjunctive graph without considering blocking constraints

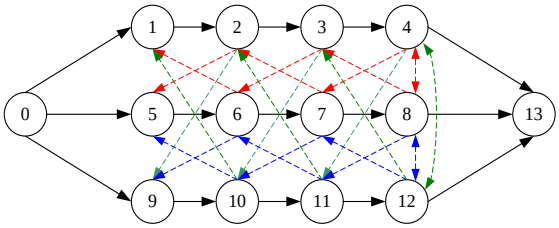

Fig.3. Transformation of the disjunctive graph to the alternative graph for considering blocking constraints

In the alternative graph in Figure 3, for each pair of operations to be executed on the same machine, there is a pair of alternative arcs replacing the pair of disjunctive arcs. For example, in the disjunctive graph in Figure 2 for operations o_1 and o_5 processed on the same machine, there is the pair of disjunctive arcs $((o_1 \to o_5),(o_5 \to o_1))$. In the alterative graph for considering the blocking conditions, the pair of disjunctive arcs $((o_1 \to o_5),(o_5 \to o_1))$ is replaced by the pair of alterative arcs $((o_{SJ[1]} \to o_5),(o_{SJ[5]} \to o_1)) = ((o_2 \to o_5),(o_6 \to o_1))$, where operation $o_{SJ[i]}$ immediately follows o_i in the same job and will be executed on a different machine. Note that a job, having completed processing on the last machine, leaves the system at once. Hence, the last machine is always available and operations o_4, o_8 and o_{12} are not blocking.

5. Feasibility Analysis

A disjunctive graph for a 2-job 2-machine job shop problem without blocking constraints is drawn in Figure 4 to explain feasibility analysis in detail.

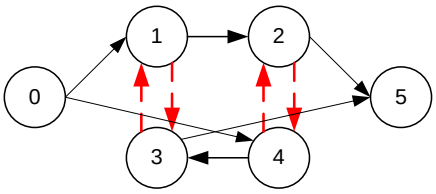

Fig. 4. The disjunctive graph for a 2-job 2-machine job shop without blocking constraints

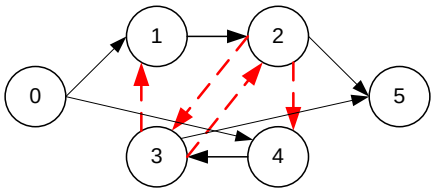

Fig. 5. The alternative graph for a 2-job 2-machine job shop with blocking constraints

In this example, each job consists of two operations, i.e. $J_1 = \{o_1, o_2\}$, $J_2 = \{o_3, o_4\}$. Job J_1 visits machine M_1 then machine M_2; Job J_2 visits M_2 then M_1. In addition, we have $M_1 = M(1) = M(3)$ and $M_2 = M(2) = M(4)$, i.e. operations o_1 and o_3 are processed on the same machine M_1; operations o_2 and o_4 are processed on the same machine M_2. If considering the blocking constraints, the two pairs of disjunctive arcs $((o_1 \to o_3),(o_3 \to o_1))$ and $((o_2 \to o_4),(o_4 \to o_2))$ are respectively replaced by the two pairs of alternative arcs $((o_2 \to o_3),(o_3 \to o_1))$ and $((o_2 \to o_4),(o_3 \to o_2))$. Thus, the corresponding alternative graph is shown in Figure 5.

For this 2-job 2-machine job shop problem with blocking conditions, we can enumerate all of four (feasible or infeasible) schedules by choosing at most one arc from each pair of alternative arcs (only two pairs for this example, $((o_2 \to o_3),(o_3 \to o_1))$ and $((o_2 \to o_4),(o_3 \to o_2))$), illustrated in Figures 6-9. For simplicity, the processing time for each operation is the same as one time unit.

1) If the alternative arcs are chosen as $(o_3 \to o_1)$ and $(o_2 \to o_4)$, the schedule is infeasible because it is cyclic (i.e. $o_1 \to o_2 \to o_4 \to o_3 \to o_1$), illustrated in Figure 6. In this case, the Gantt chart cannot be drawn because the schedule is cyclic or infeasible.

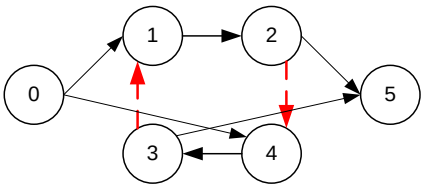

Fig. 6. One infeasible schedule of a blocking 2-job 2-machine job shop with selected alternative arcs $(o_3 \to o_1)$ and $(o_2 \to o_4)$

2) If the choosing alternative arcs are $(o_3 \to o_1)$ and $(o_3 \to o_2)$, the schedule with its corresponding Gantt chart are presented in Figure 7.

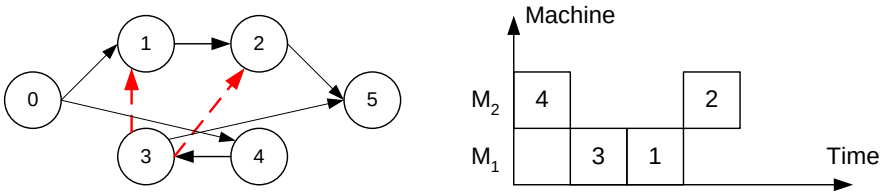

Fig. 7. One feasible schedule of a blocking 2-job 2-machine job shop with selected alternative arcs $(o_3 \to o_1)$ and $(o_3 \to o_2)$

3) If the choosing alternative arcs are $(o_2 \to o_3)$ and $(o_2 \to o_4)$, the schedule is shown in Figure 8.

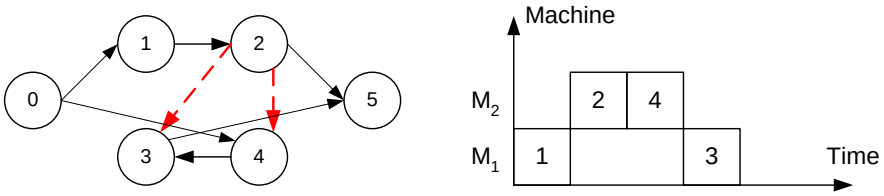

Fig. 8. One feasible schedule of a blocking 2-job 2-machine job shop with selected alternative arcs $(o_2 \to o_3)$ and $(o_2 \to o_4)$

4) If the choosing alternative arcs are $(o_2 \to o_3)$ and $(o_3 \to o_2)$, whether the schedule is feasible depends on the particular context, described in Figure 9 and analysed in the following.

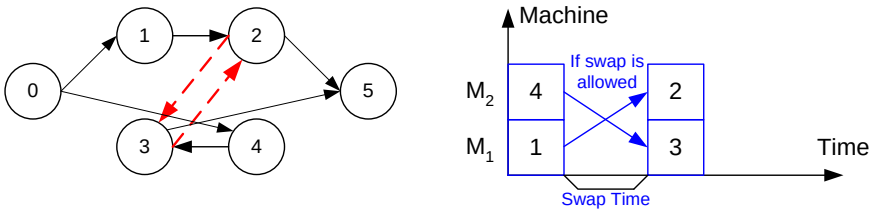

Fig. 9. One (feasible or infeasible) schedule of a blocking 2-job 2-machine job shop with selected alternative arcs $(o_2 \to o_3)$ and $(o_3 \to o_2)$; note that the schedule is infeasible if swap is not allowed and the Gantt Chart can be drawn only for swap-allowed blocking case.

In such a situation in Figure 9, we need to distinguish two cases of blocking: *swap-allowed blocking* and *no-swap blocking*. From the alternative graph, by selecting $(o_2 \to o_3)$ and $(o_3 \to o_2)$,

we obtain an cycle $(o_2 \to o_3 \to o_2)$ or $(M(2) \to M(3) \to M(2))$. In such a situation, all jobs in the cycle must move simultaneously to the next machine in a cycle. This blocking situation is called "***deadlock***". To be feasible, a "***swap***" manipulation is needed whenever there is a cycle of two or more jobs, each one waiting for a machine which is blocked by another job in the cycle. It is intuitive that, depending on the particular context, a swap may be allowed or not. The swap is allowed when the jobs can move independently of each other. On the contrary, the swap is not allowed when the jobs can move strictly after that the subsequent resource becomes available. The deadlock situation is similar to a conflict when one outbound train and one inbound train are crossing in the single-track section. For safety, the deadlock situation is strictly prohibited in the train scheduling problems, namely, the train schedule should be deadlock-free. As the swap is not allowed for blocking trains, the deadlock-freeness conditions may be guaranteed only when the resources are available in multiple units (i.e. parallel machines or multiple-track sections). More specially, the deadlock and deadlock-freeness situations are respectively illustrated in Figure 10.

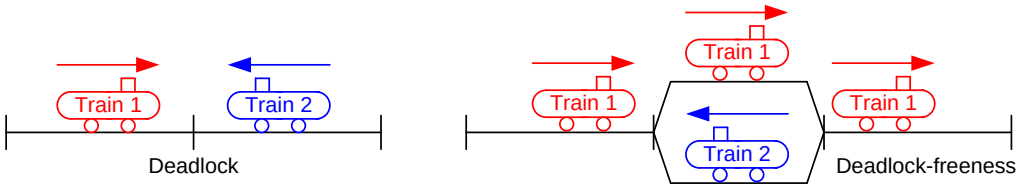

Fig. 10. Illustration of deadlock and deadlock-freeness situations in train scheduling

6. Solution Techniques for BPMJSS

It is well known that the idea of shifting bottleneck procedure (SBP) is initiative as a solution to the classical JSS problem [16]. This is due to the analysis that a processing order of operations (i.e. job sequence) on one machine is equivalent to an acyclic selection of disjunctive arcs that contains exactly either of each pair of disjunctive arcs on this machine. Consequently, the machines can be considered one by one, and the results are applied to both rank the machines and select a good sequence on the highest rank machine (i.e. bottleneck machine).

First, without considering blocking conditions, a parallel-machine job-shop-scheduling (PMJSS) problem is solved by an improved SBP with four improvements, summarised as below.

- the topological-sequence algorithm [17-20] is applied to solve the partial graph model and then decompose PMJSS into a set of single-machine-scheduling (SMS) and (or) parallel-machine-scheduling (PMS) subproblems;
- a modified Carlier algorithm [21] based on our proposed theorems is developed for polynomially solving the SMS subproblems;
- an extended algorithm based on Jackson rules [22] is implemented to polynomially solve the PMS subproblems;
- *Tabu Search* and *Simulated Annealing* metaheuristics are combined to improve the partial sequence in the step of re-sequencing and re-optimisation iterations of SBP. This hybrid algorithm is also used to further optimise the complete sequence.

Inspired by SBP algorithm for PMJSS, in order to consider blocking conditions, we then propose an innovative algorithm for BPMJSS. This algorithm is called *Feasibility Satisfaction Procedure* (FSP) which can schedule trains consecutively one at a time. The architecture of the proposed FSP algorithm for BPMJSS is described as below.

- Apply developed SBP algorithm to solve the PMJSS model.
- Check whether the obtained schedule by SBP is feasible, i.e. satisfying all the blocking constraints.

- If feasible, add the new train and update the input data; apply SBP algorithm again to obtain new schedule and check feasibility again.
- If infeasible, load the feasible graph model generated at previous (or initial) step; apply Insertion Algorithm (see below) to insert the operations of the added train respectively to each single-track section and multiple-track sections and satisfy the blocking constraints; then check whether this new graph model is feasible.
- Apply "Feasibility Satisfaction Procedure" iterations till the feasible complete graph model is obtained.

In Feasibility Satisfaction Procedure, we design an insertion algorithm which can effectively insert a new train into the given graph to generate a new graph satisfying the blocking conditions consecutively. In insertion algorithm, it is very important to construct an efficient data structure that can conveniently track all the necessary information for analysing blocking conditions, including the direction of trains, sectional running times of each train, the earliest available time of single-track and/or multiple-track sections, the ready time, the starting time, the completion time, the blocking time, the departure time of inserted train's operation, and etc. More specially, given a partial feasible schedule u^k at iteration k of Feasibility Satisfaction Procedure (FSP) and one train T_{k+1} to be added, the basic steps of insertion algorithm is described as follows.

Step 1: Let s denote one track section and operation $O_{T_{k+1},s}$ denote the traversal of train T_{k+1} on s.

Step 2: If s is single-track section, then go to Step 3; else if s is multiple-track section, then go to *Step 4.*

Step 3: Assume that the train list of u^k on s is $(O_{T_1,s}, O_{T_2,s}, \ldots, O_{T_k,s})$, to find the best insertion point while to satisfy the blocking conditions, we consider the following three choices in order:
i) whether $O_{T_{k+1},s}$ can be inserted before $O_{T_1,s}$;
ii) whether $O_{T_{k+1},s}$ can be inserted in the middle between $O_{T_1,s}$ and $O_{T_k,s}$ as early as possible; and
iii) otherwise, insert $O_{T_{k+1},s}$ after $O_{T_k,s}$.

Step 4: As s is multiple-track section, assume the number of track units is n_s and $s=\{s_{u_1}, s_{u_2}, \ldots, s_{u_{n_s}}\}$. In addition, each unit is associated with the train's direction (outbound or inbound), i.e. $s=\{s_{u_1}, s_{u_2}, \ldots, s_{u_{n_s}}\}=\{s_{inbound}\}\oplus\{s_{outbound}\}$. If the direction of train T_{k+1} is inbound, consider to insert $O_{T_{k+1},s}$ on one unit of a multiple-track section s_{u_i} ($s_{u_i}\in\{s_{inbound}\}$) having the earliest available time. In this way, s_{u_i} is treated as a single-track section. If the direction of train T_{k+1} is outbound, it is in the same fashion but with $s_{u_i}\in\{s_{outbound}\}$.

Step 5: Go to *Step 1* with considering the next adjacent section. If all the operations of train T_{k+1} are inserted, thus u^k is updated to u^{k+1} and iteration $k+1$ of FSP is finished.

Furthermore, with satisfying the constraints on blocking conditions and limited capacity of resources, the time-determination procedure for one operation (the movement/traversal of a train across a section) in the above *Steps 3* and *4* is the core of the FSP algorithm. The pseudocode of time-determination procedure with key formulae is presented in Appendix 1.

7. Computational Experiment

The following computational experiment of a real-world train scheduling case is used to compare and analyse the PMJSS and BPMJSS models.

In this example, six outbound trains and four inbound trains are traversing on 10 single-track sections and 9 double-track sections. For simplicity, in this train scheduling case, the sectional running times for different trains on the same section are identical and the additional running time

caused by train's length is temporarily included in the sectional running time. At first, without considering the blocking constrains, this case is modelled as a PMJSS problem and solved by the improved SBP algorithm. The proposed methodology is coded by Visual C++ in MFC environment. All typical application-specific modules are treated as the following objects: data, formulae, algorithms, output, graphics and other behaviours. Thus, the object-oriented programming tends to produce software that is more understandable and better organised. For example, it provides a convenient way to quickly obtain the new feasible solutions for different type scheduling problems by changing the attributes in data input. And the solutions can be conveniently shown and analysed in the graphic interface.

Firstly, the obtained PMJSS result for this test problem is displayed by the Gantt chart shown in Figure 11, in which different trains are distinguished by the colour and the number.

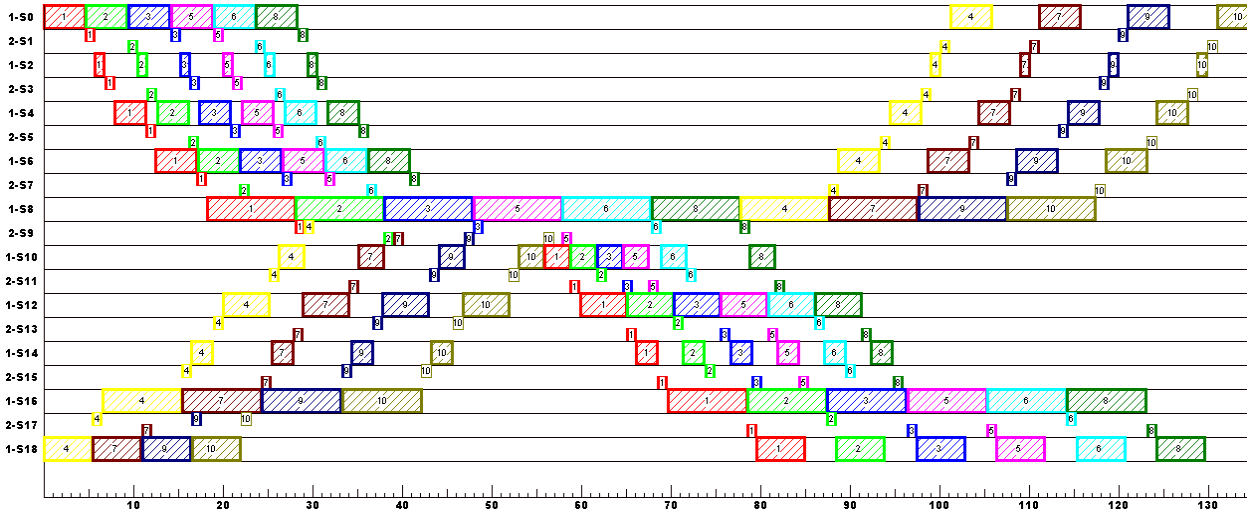

Fig. 11. The Gantt chart for the PMJSS schedule obtained by SBP

This PMJSS result is feasible if the storage capacity between successive machines is unlimited, i.e. without considering blocking conditions. However, it becomes infeasible if this PMJSS schedule is implemented for train scheduling environment, for which the real-life case should consider hold-while-wait (or blocking) constraints and deadlock situations. For illustrating infeasibility, this PMJSS schedule is depicted in the String chart, as shown in Figure 12, for describing the relationship between train's position and time point. It is easy to indicate from the String chart that there are many train conflicts (overlaps) in double-track Sections 2-S7, 2-S9, 2-S11, 2-S15, and 2-S17. In a real-life train scheduling case, the situation that more than one train remain on a track section at the same time is very dangerous and definitely prohibited.

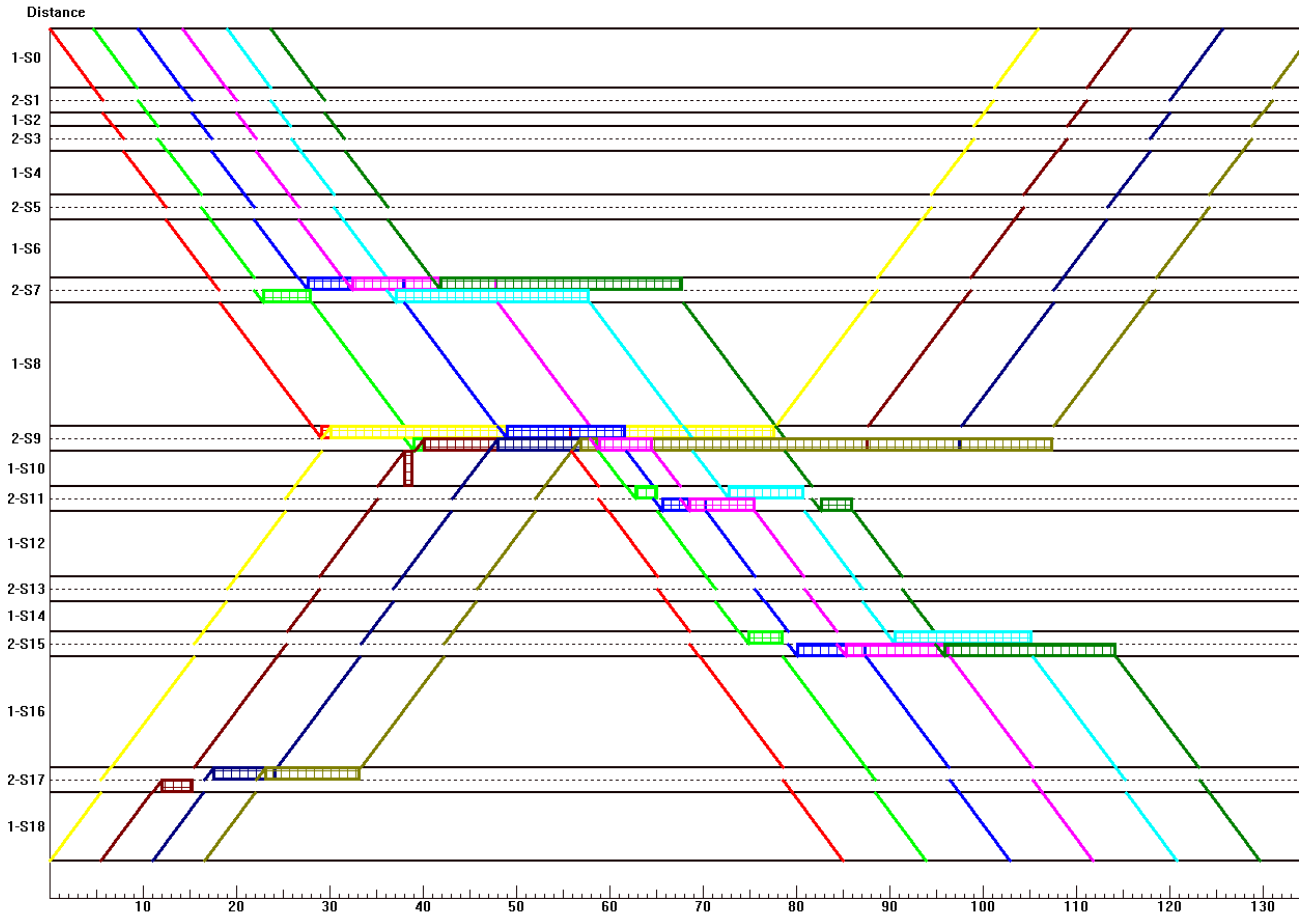

Fig. 12. The String chart for the PMJSS schedule obtained by SBP, in which the blocking conditions are highlighted by cross brush

Therefore, the train scheduling problem has to be modelled as a BPMJSS problem in alternative graph for considering the blocking conditions. The proposed BPMJSS model can be solved by a new algorithm mentioned above, i.e. *Feasibility Satisfaction Procedure* (FSP). The Gantt chart of the BPMJSS schedule obtained by FSP for this test case is drawn in Figure 13.

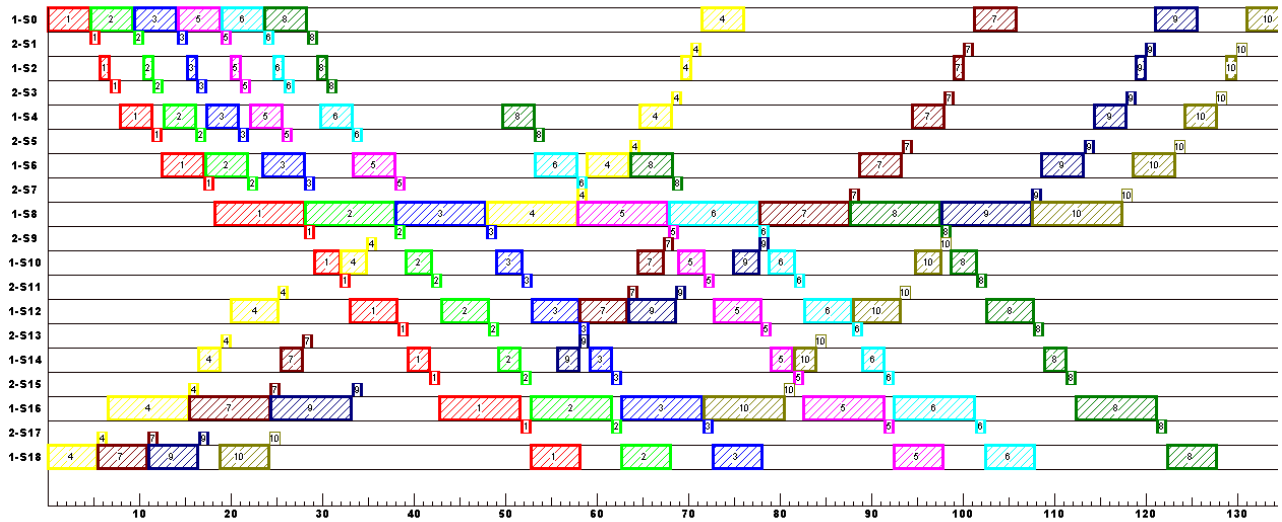

Fig. 13. The Gantt chart for the BPMJSS schedule obtained by FSP

For the sake of analysing feasibility, the String chart for this BPMJSS schedule obtained by FSP is given in Figure 14. As there is not any overlap of cross-brush box in Figure 14, we can say that

the blocking constraints are satisfied and this BPMJSS schedule may be feasible or applicable for real-world implementation.

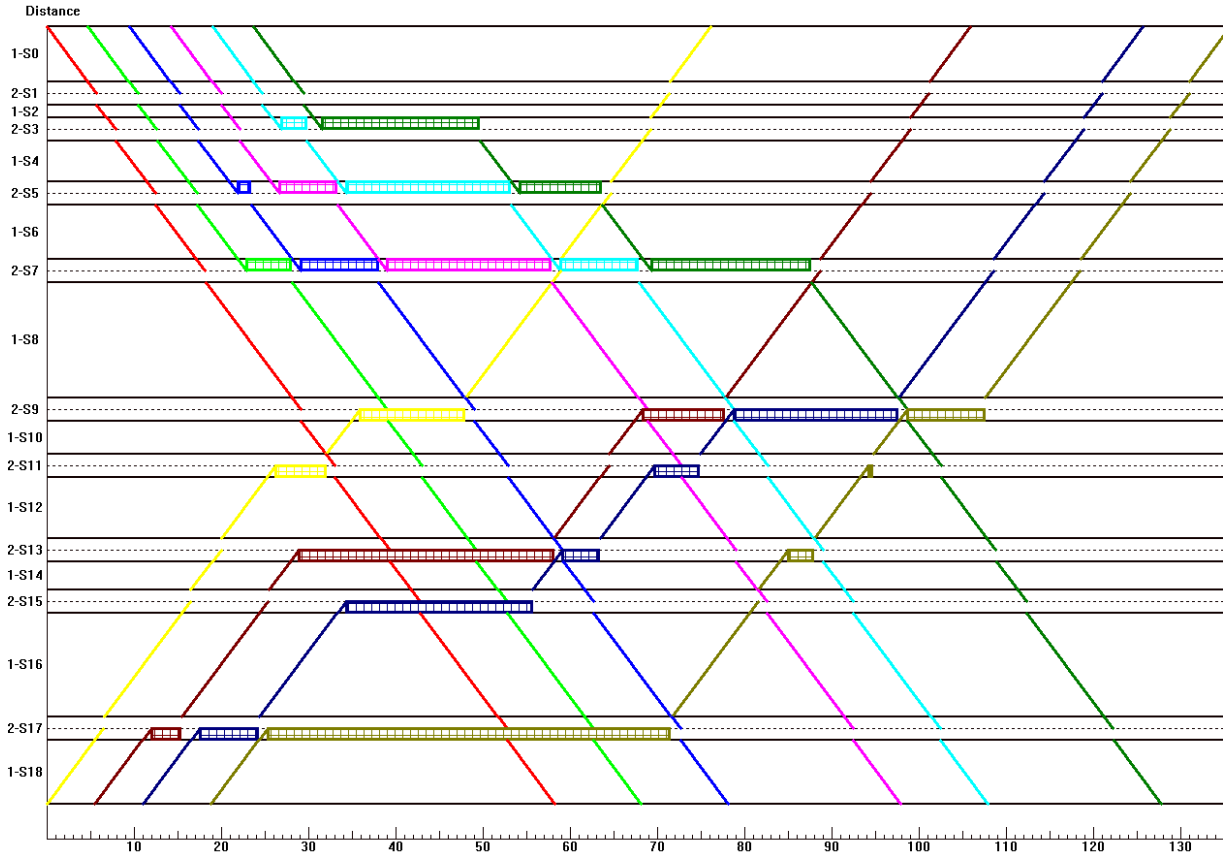

Fig. 14. The String chart of the BPMJSS schedule obtained by FSP, in which the blocking conditions are highlighted by the cross brush

8. Implementations

In this section, some real-world implementations of the proposed methodology are discussed.

i) Considering the train length

First, in practice, the train length should be seriously considered because it has a great effect on the performance of operating a railway. This is because, when a train is traversing from one section into the next section, the train has to occupy these two sections in a period (i.e. the occupying time caused by train length) till the whole body of the train completely leaves the section. The analysis of effects caused by the train length is given in Figure 15.

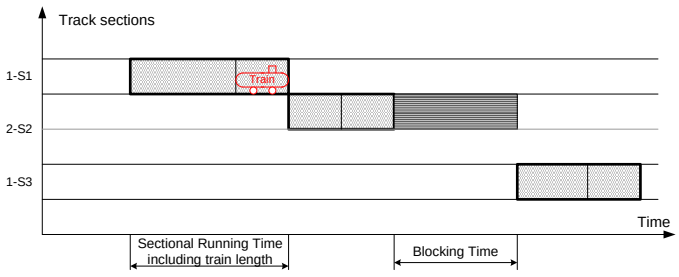

Fig.15.(a) The analysis when the sectional running time includes train length

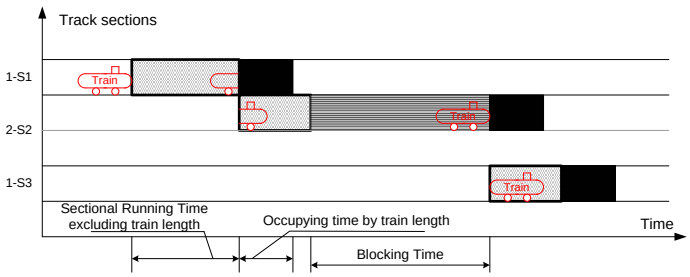

Fig.15.(b) The analysis when the sectional running time excludes train length

As observed from Figure 15(b), when the sectional running time excludes the occupying time caused by train length, the starting time and completion time of the operation on Section 2-S2 become smaller in comparison. The below new parameters are adopted to obtain the more realistic solutions when train length is seriously considered.

Ω_i	occupying time of operation i, caused by the corresponding train length
p_i'	new sectional running time of operation i ($p_i' = p_i - \Omega_i$)
c_i'	new completion time of operation i when the sectional running time excludes train length ($c_i' = c_i - \Omega_i$)
$SJ[i]$	the same-job successor of operation i
$SM[i]$	the same-machine successor of operation i
$e'_{SJ[i]}$	new starting time of operation $SJ[i]$
b_i'	new blocking period of operation i ($b_i' = e'_{SJ[i]} - c_i'$)
D_i'	new departure time ($D_i' = c_i' + b_i'$)
L_i	leave time at which the whole body of the train completely enters the next section ($L_i = D_i' + \Omega_i$)
$e'_{SM[i]}$	new starting time of operation $SM[i]$ ($e_{SM[i]} = \max(e_{SJ[i]}, L_{PJ[SM[i]]}, L_i)$)

To further verify the great effects caused by train length, this more realistic BPMJSS case when the occupying times (measured as 0.5 times units for all trains) caused by train length are excluded from the sectional running times for each train, is solved by the FSP algorithm with the above adjusted formulae. The corresponding result is displayed in Figure 16. It is observed that the makespan decrease from 135.78 in Figure 13 to 127.28 in Figure 16, although the train schedule in Figure 16 is kept the same as that in Figure 13.

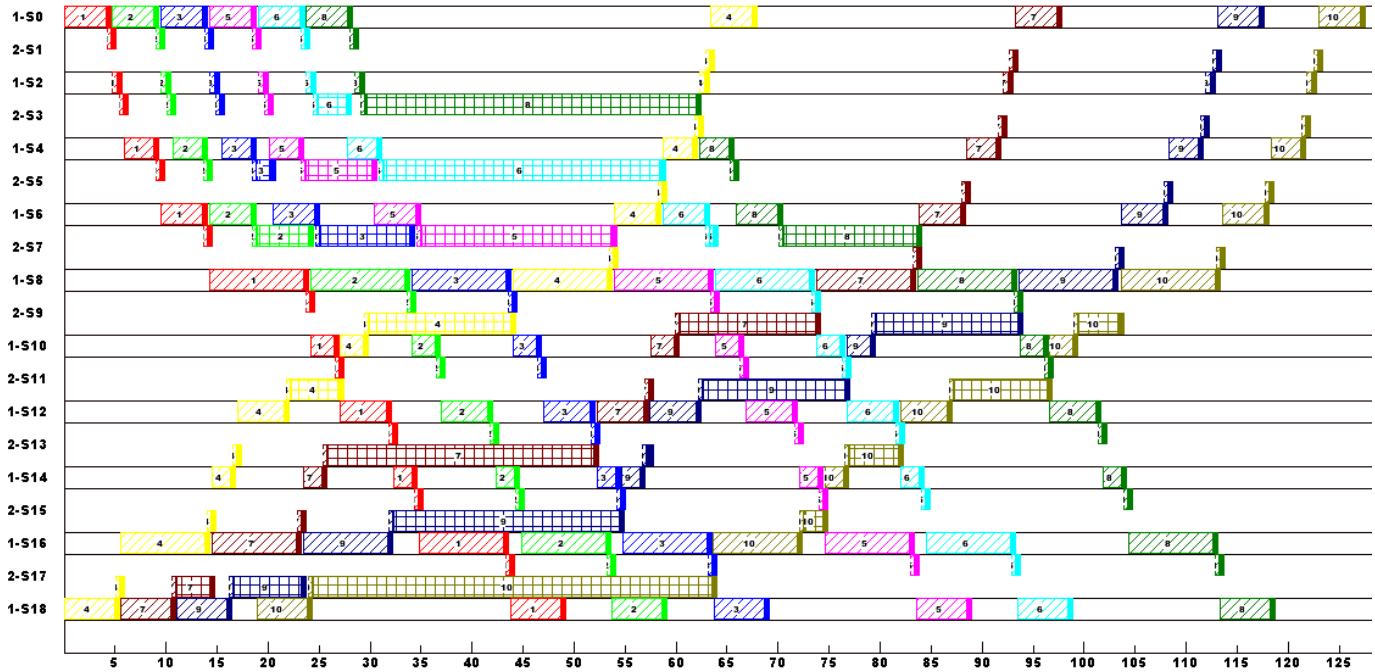

Fig.16. The Gantt chart for the new BPMJSS result when the sectional running time excludes train length, in which the occupying times are highlighted by the solid brush

ii) Upgrading track sections

In addition, the proposed methodology is very helpful for the decision making on upgrading the number and location of track sections. As seen in Figure 13, for example, the single-track section 1-S8 is called a *bottleneck section* because it has the minimum number of gaps (idle times) between trains. If this bottleneck section (1-S8) and its adjunct double-track sections (2-S7 and 2-S9) are upgrading by respectively adding one more track. In this case, the new makespan declines to 108.03 in Figure 17 from 135.78 in Figure 13. This is a huge improvement on efficiency of operating the overall railing system.

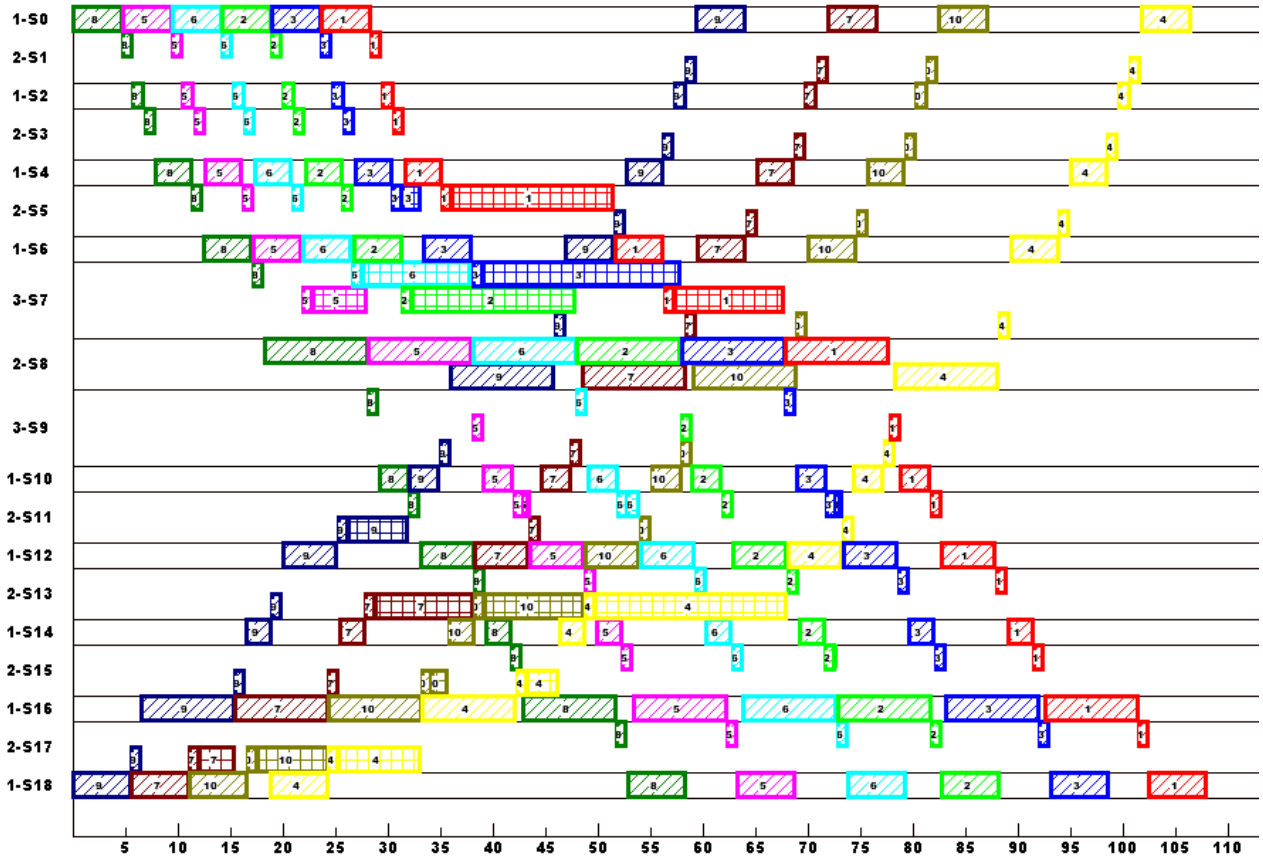

Fig. 17. The Gantt chart of the BPMJSS schedule obtained by FSP, in which Sections 1-S8, 2-S7, 2-S9 are upgrading by adding one more track

iii) Increasing train speed of one tardy train on a critical section

Moreover, the proposed methodology is suitable for dynamic scheduling. For example, assuming that Train 4 in Figure 17 arrives late at the destination, there is an easy way to make Train 4 arrive on time while the timetables of other trains are unchanged. This method only needs to increase the speed of Train 4 on Section 1-S12 in such that the sectional running time is smaller than the interval time between Train 6 and Train 2 on this section. Thus, the new result can be easily obtained by applying the FSP algorithm to the updated data. As a result, the arrived time of Train 4 at destination is decreased from 106.51 in Figure 17 to 97.14 in Figure 18.

From the Gantt chart, it seems to be essential to identify the so-called *critical section* in order to let tardy train arrive on time. In this example, for Train 4 in Figure 17, one of critical sections is

Section 1-S12. It is observed that one tardy train can arrive early only by increasing the train speed on a critical section, without adjusting the timetables of any other trains.

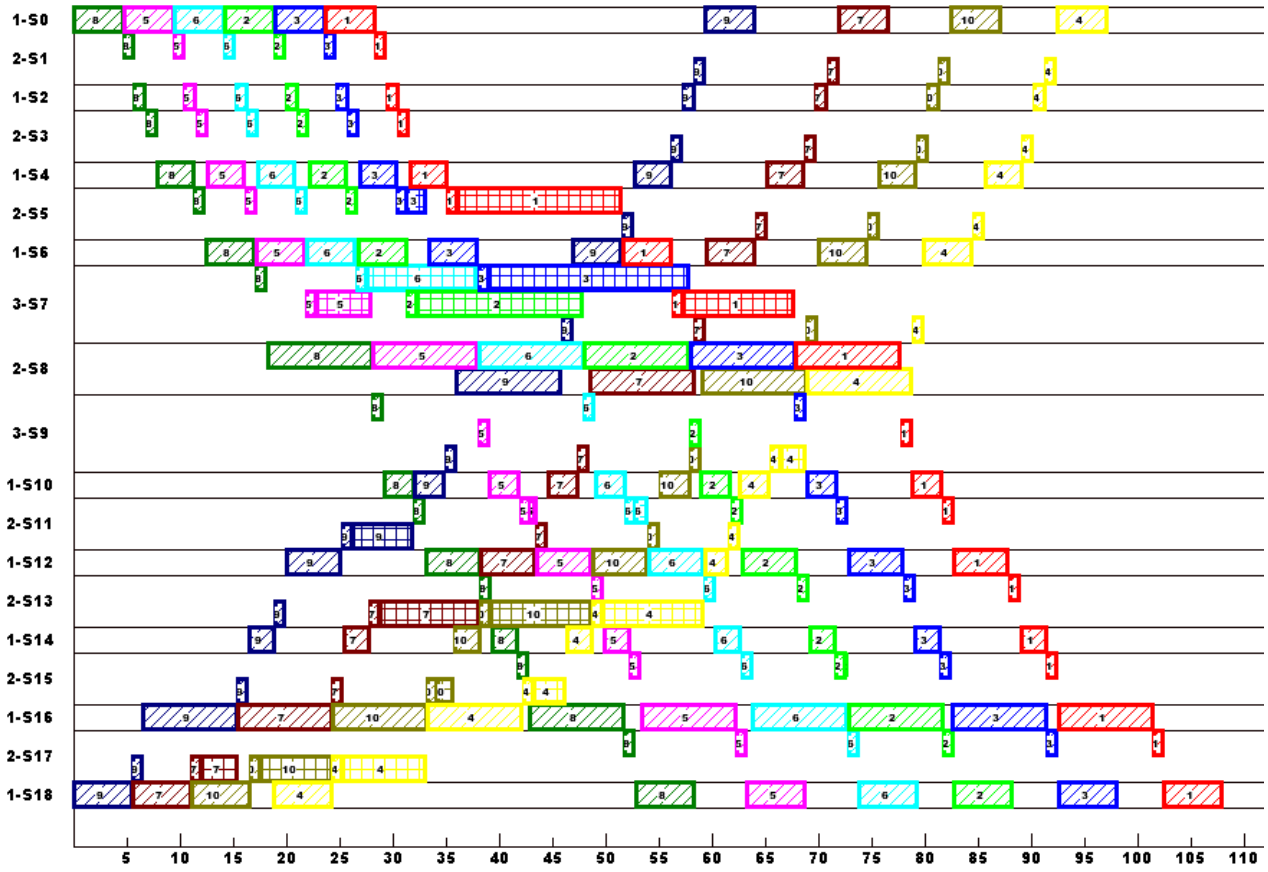

Fig. 18. The new BPMJSS result when the train speed of Train 4 increases on a critical section (1-S12)

iv) Shorten sectional running times of all trains on a bottleneck section

Furthermore, in many real-life situations, it is nearly impossible to upgrade the railway infrastructure especially in residential or tunnel areas due to extremely high cost. In this case, without any investment in expanding railway facilities, it is still able to improve the railing capacities and efficiency by shortening sectional running times of all trains on the so-called *bottleneck section* on which it has the minimum number of gaps between trains. This phenomenon can be validated by the below results.

If the sectional running times of all trains on the bottleneck section (i.e. 1-S8 in Figure 13) are shorten to be approximately a half, it is observed from the new result shown in Figure 19 that the makespan drops to 110.10, compared to 135.78 in Figure 13. In addition, it is noted in Figure 19 that the bottleneck section is shifted from Section 1-S8 in Figure 13 to Section 1-S16.

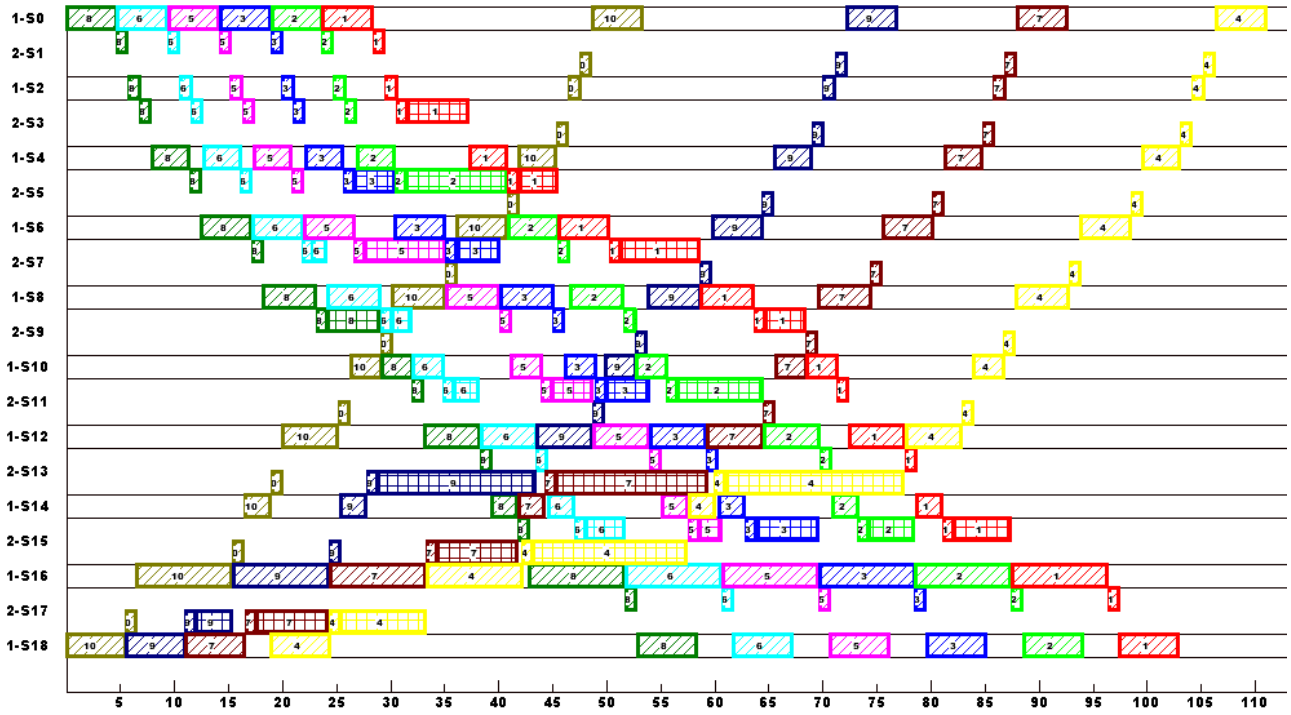

Fig. 19. The new BPMJSS result when the sectional running times of all trains on Section 1-S8 are shorten to be approximately a half; it is noted that the new bottleneck section is Section 1-S16

Next, if the sectional running times of each train on the new bottleneck section (i.e. 1-S16 in Figure 19) become approximately a half, the makespan continuously falls to 96.56, as shown in Figure 20.

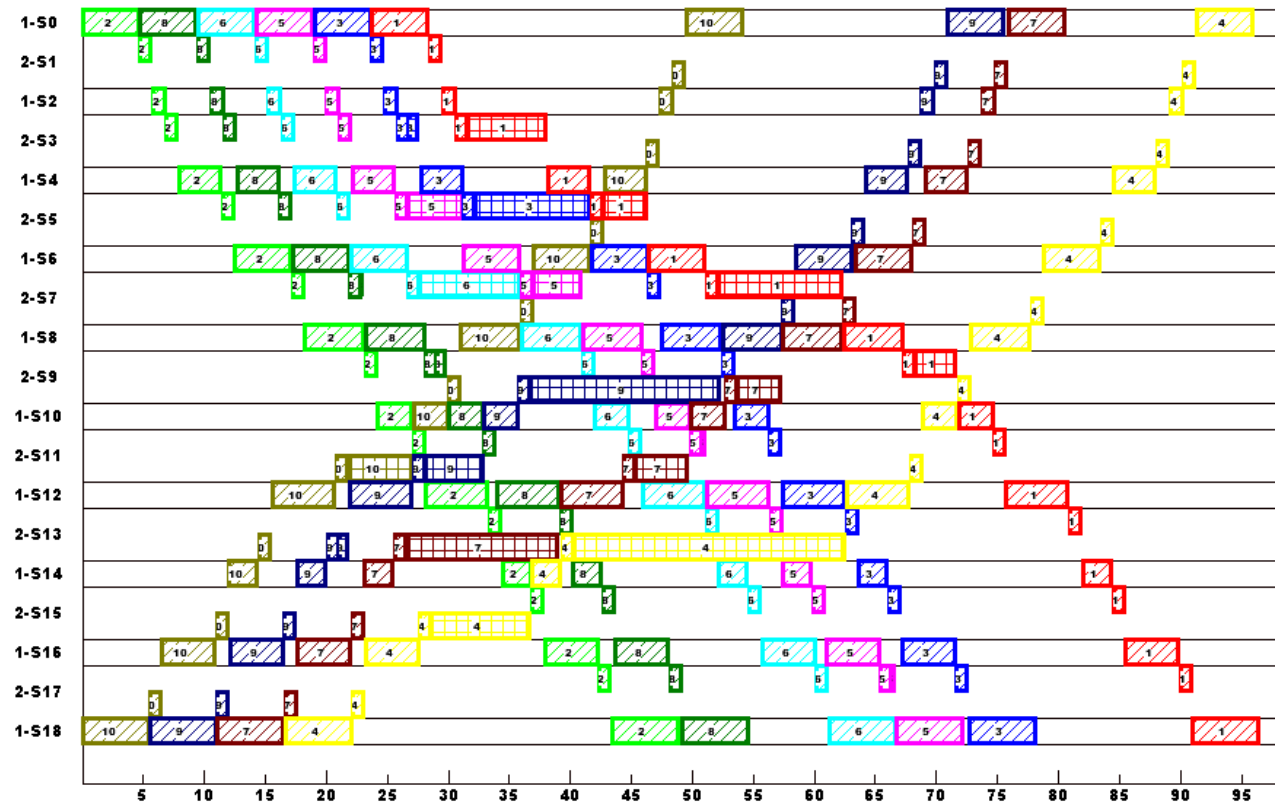

Fig. 20. The new BPMJSS result when the sectional running times of all trains on two bottleneck sections (i.e. Section 1-S8 and Section 1-S16) are shorten to be approximately a half at the same time

9. Conclusions

In this paper, the train scheduling problem is modelled as a *blocking parallel-machine job-shop-scheduling* (BPMJSS) problem. Moreover, the BPMJSS problem is formulated and analysed in the alternative graph that is an extension of the classical disjunctive graph. Based on our analysis and observation, it is not trivial to directly find a feasible complete BPMJSS schedule. Inspired by *Shifting Bottleneck Procedure* for PMJSS, we propose a new constructive heuristic algorithm called *Feasibility Satisfaction Procedure* (FSP) to obtain the feasible BPMJSS schedule. A real-world train scheduling case is studied and illustrated for comparing and analysing the PMJSS and BPMJSS models. Some real-life applications including considering the train length, upgrading the track sections, increasing train speed of one tardy train on the critical section and shortening the sectional runtime times of all trains on bottleneck sections are discussed.

The proposed methodology would be very promising because it can be applied as a fundamental tool for modelling and solving many real-world scheduling problems that should consider the capacity of resources (machines or track sections) and different inter-resource buffer conditions. In addition, the BPMJSS model can be extended to be a *No-Wait Blocking Parallel-Machine Job-Shop Scheduling* (NWBPMJSS) problem for modelling complicated overtaking and crossing situations in train scheduling problems, in which freight (blocking) trains and passenger (no-wait or continuous) trains are considered simultaneously. No-wait conditions arise when considering the passenger trains that should traverse continuously without any interruption and any unplanned dwelling, for which the customer has a prohibitively high cost of waiting in traversing. Thus, passenger trains are treated as no-wait (continuous) trains. In comparison, freight trains are allowed to enter the next section immediately if possible or to remain in a section until next section on the routing becomes available, which is thought of as a relaxation of no-wait condition that is stricter.

Further research is needed in order to improve the solution quality of BPMJSS schedule. Different possibilities include developing sophisticated neighbourhood search guided by well-known metaheuristics such as *Tabu Search*, or embedding the proposed FSP algorithm as a generic base heuristic into the framework of the look-ahead metaheuristics, similar to *Rollout Algorithm* [23] or *Pilot Method* [24].

Acknowledgments

One part of this paper has been presented at the 8th Asia Pacific Industrial Engineering & Management Systems Conference. Financial support from Queensland University of Technology and the data provided by Queensland Rail for this study are gratefully acknowledged.

Appendix 1

Initialise the given information and data structure of one operation, including train (job) index, section (machine) index, number of section units (default as 1 for single-track section), train direction, operation index, and processing time (*PTime*).
If the direction of this train is inbound,
  If the section is a single-track section,
    Set the index of the same-job predecessor: *PJOper*.
    Set the completion time of the same-job predecessor: *CTime_PJOper*.
    Set the earliest available time of the next section: *ATime_NextMachine*.
    Set the ready time of this operation:
      RTime = max(*ATime_NextMachine*−*PTime*, *CTime_PJOper*).
    Set the starting time of the first scheduled operation on this section: *ETime_FOper*.

Set the interval time between *RTime* and *ETime_FOper*: *ITime_F = ETime_FOper−RTime*.
If *ITime_F >= PTime*, the starting time of this operation equals ready time:
ETime = RTime.
Else set the number of operations that have been scheduled on this section, *nSOpers*.
For the first scheduled operation to the last second scheduled operation on this section,
Set the departure time of the previous operation (e.g. the first scheduled operation) and the starting time of its successive operation (e.g. the second scheduled operation):
DTime_PreOper, *ETime_SucOpe*.
Set the interval time:
ITime_M = min(*ETime_SucOper−DTime_PreOper, ETime_SucOper−RTime*).
If *ITime_M >= PTime*, the starting time of this operation is set as:
ETime = max(*DTime_PreOper*, *RTime*);
break.
If *ETime* is still not determined, set the departure time of the last scheduled operation on this machine: *DTime_LOper*; and the starting time of this operation is set as:
ETime = max(*DTime_LOper*, *RTime*).
Set the blocking time of the same-job predecessor:
BTime_PJOper = max(0, *ETime− CTime_PJOper*).
Set the departure time of the same-job predecessor:
DTime_PJOper = CTime_PJOper + BTime_PJOper.
If the section is a multiple-track section,
Set the number of the multiple-track section units for traversing inbound trains: *nInUnits*.
Among them, determine the unit index with its earliest available time: *ThUnit* and *ATime*.
The starting time of this operation is set as:
ETime = max(*CTime_PJOper*, *ATime*).
If the direction of this train is outbound, most of the above formulae are used in the same fashion but in a reverse direction. The unique difference is to determine the unit index with the earliest available time among the multiple-track section units for outbound trains.

References

[1] Higgins A, Ferreira L, Kozan E. Modelling single line train operations. Transportation Research Record, Journal of the Transportation Research Board, Railroad Transportation Research 1995; 1489: 9-16.

[2] Abdekhodaee A, Dunstall S, Ernst AT, Lam L. Integration of stockyard and rail network: a scheduling case study. Paper presented at the Proceedings of the Fifth Asia Pacific Industrial Engineering and Management Systems Conference, Gold Coast, Australia, 2004.

[3] Kozan E, Burdett RL. A railway capacity determination model and rail access charging methodologies. Transportation Planning and Technology, 2005; 28(1): 27-45.

[4] Burdett R L, Kozan E. Techniques for absolute capacity determination in railways. Transportation Research Part B, 2006; 40: 616-632.

[5] Ingolotti L, Tormos P, Lova A, Barber F, Salido MA, Abril M. A decision supporting system (DSS) for the railway scheduling problem. Unpublished manuscript, 2005.

[6] Epstein DJ, Lu Q, Zhao J, Leachman, RC. An exact solution procedure for determining the optimal dispatching times for complex rail networks. Technical report, 2005; Department of Industrial and Systems Engineering, University of Southern California, Los Angeles, CA, USA.

[7] Lindner T, Zimmermann UT. Cost Optimal Periodic Train Scheduling. Technical Report, 2005; Institut fur Mathematische Optimierung, Technische Universitat Braunschweig.

[8] Zhou X, Zhong M. Bicriteria train scheduling for high-speed passenger railway planning applications. European Journal of Operational Research, 2004; 167: 752-771.

[9] Higgins A, Kozan E, Ferreira L. Heuristic techniques for single line train scheduling. Journal of Heuristics, 1997; 3: 43-62.
[10] Higgins A., Kozan E. Modeling Train Delays in Urban Networks. Transportation Science, 1998; 32(4), 346-357.
[11] Oliveira E, Smith BM. A job-shop scheduling model for the single track railway scheduling problem: Research Report Series, 2000; School of Computing, University of Leeds.
[12] Chew KL, Pang J, Liu QZ, Ou JH, Teo CP. An optimization based approach to the train operator scheduling problem at Singapore MRT. Annals of operations Research, 2001; 108(1): 111-118.
[13] Pacciarelli D, Pranzo M. A tabu search algorithm for the railway scheduling problem. Paper presented at the MIC'2001 - 4th Metaheuristics International Conference, 2001; 159-165.
[14] Roy B, Sussmann B. Les Problèmes d'ordonnancement avec contraintes disjonctives. Note DS n.9 bis, SEMA, Montrouge, 1964.
[15] Masics A, Pacciarelli D. Job-shop scheduling with blocking and no-wait constraints. European Journal of Operational Research, 2002; 143: 498-517.
[16] Adams J, Balas E, Zawack D. The shifting bottleneck procedure for job shop scheduling. Management Science, 1988; 34(3): 391-401.
[17] Liu SQ, Ong HL. A comparative study of algorithms for the flowshop scheduling problem. Asia-Pacific Journal of Operational Research, 2002; 19: 205-222.
[18] Liu SQ, Ong HL. Metaheuristics for the mixed shop scheduling problem. Asia-Pacific Journal of Operational Research, 2004; 21(4): 97-115.
[19] Liu SQ, Ong HL, Ng KM. Metaheuristics for minimizing the makespan of the dynamic shop scheduling problem. Advances in Engineering Software, 2005a; 36: 199-205.
[20] Liu SQ, Ong HL, Ng KM. A fast tabu search algorithm for the group shop scheduling problem. Advances in Engineering Software, 2005b; 36: 533-539.
[21] Carlier J. The one-machine sequencing problem. European Journal of Operational Research, 1982; 11: 42-47.
[22] Jackson JR. Scheduling a production line to minimize maximum tardiness. Technical Report 43. Los Angeles: University of California, 1955.
[23] Bertsekas DP, Tsitsiklis JN, Wu C. Rollout algorithms for combinatorial optimization problems. Journal of Heuristics, 1997; 3: 245-262.
[24] Voß S, Fink A, Duin C. Looking ahead with the pilot method. Annals of Operations Research, 2005; 136: 285-302.