Queensland University of Technology
Brisbane Australia

This is the author's version of a work that was submitted/accepted for publication in the following source:

Burdett, Robert L. & Kozan, Erhan (2009) A sequencing approach for creating new train timetables. *OR Spectrum*, *32*(1), pp. 163-193.

This file was downloaded from: http://eprints.qut.edu.au/29827/

**© Copyright 2009 Springer**

**Notice**: *Changes introduced as a result of publishing processes such as copy-editing and formatting may not be reflected in this document. For a definitive version of this work, please refer to the published source:*

*http://dx.doi.org/10.1007/s00291-008-0143-6*

# A Sequencing Approach for Creating New Train Timetables

**R. L. Burdett and E. Kozan**

School of Mathematical Sciences, Queensland University of Technology,
PO Box 2434, QLD 4001, Australia

**Abstract:** Train scheduling is a complex and time consuming task of vital importance. To schedule trains more accurately and efficiently than permitted by current techniques a novel hybrid job shop approach has been proposed and implemented. Unique characteristics of train scheduling are first incorporated into a disjunctive graph model of train operations. A constructive algorithm that utilises this model is then developed. The constructive algorithm is a general procedure that constructs a schedule using insertion, backtracking and dynamic route selection mechanisms. It provides a significant search capability and is valid for any objective criteria. Simulated Annealing and Local Search meta-heuristic improvement algorithms are also adapted and extended. An important feature of these approaches is a new compound perturbation operator that consists of many unitary moves that allows trains to be shifted feasibly and more easily within the solution. A numerical investigation and case study is provided and demonstrates that high quality solutions are obtainable on real sized applications.

**Keywords:** Job shop scheduling, train timetabling, meta-heuristics

## 1. Introduction

Trains provide a relatively clean and cheap method of transportation for passengers and freight, and compare favourably if not better than alternative modes of transportation such as road, air and sea in many circumstances. Furthermore the utilisation of railway systems can only increase in the future as roads become even more congested, trains become faster and infrastructure is extended. Due to the size, weight and speed of trains the coordination of train movements (by train scheduling) is vital in order to utilise these systems safely and effectively. However train scheduling on current systems is still a relatively difficult and time consuming task as the size and complexity is prohibitive. Train scheduling problems have unique properties and pose a number of unique difficulties that distinguish it from other related scheduling problems. These will be discussed in a later section. The manual construction of a schedule by a human expert with the help of computer software is the most common first and last resort in practice.

In practice there are a variety of different scheduling problems that must be solved, though in principle two main variants exist. The first considers the development of a new timetable that is typically but not necessarily to be applied at regular intervals such as daily, weekly or monthly. The second scheduling problem concerns the re-development of an existing timetable. For example, an existing timetable may become undesirable and or infeasible after unforseen delays have caused significant deviations to the original plan. In the first variant there is usually no limitation on when trains may enter the system, i.e. they may enter at any time. However in the second variant trains have to enter at predefined time and some trains may already be within the system at the start of the schedule. In recent years the majority of papers in the literature have addressed the second "rescheduling" problem, and examples include Carey M. (1994a,b) and Higgins et al (1996) for exact approaches and Cai

and Goh (1994), Higgins and Kozan (1997), Cai et al (1998), Chiang et al (1998), Sahin I. (1999), and Adenso-Diaz et al (1999), Dorfman and Medanic (2004) for heuristic approaches. The first problem has been addressed more recently by Odijk (1996), Brannlund et al (1998), Goverde (1999), Lindner (2000), Kroon and Peeters (2003). Train platforming and pathing is another aspect that has received attention recently by Carey and Lockwood (1992), Zwaneveld et al (1996), Kroon et al (1997), Cordeau et al (1998), Zwaneveld et al (2001), Billionnet (2003), Carey and Carville (2003).

In this paper the most efficient way for a specified number (mix) of trains with predefined routes to traverse a railway system (network) between their pre-defined origin and destination location subject to a variety of technical constraints is considered. This is achieved by a new "hybrid" job shop scheduling (JSP) approach. A job shop approach is taken as it is new and more importantly because it has the potential to be significantly better than other existing approaches. To our knowledge this approach has not been taken before and if so, not to the same extent to which it is taken in this paper. In recent years however some aspects of the train scheduling problem have been addressed separately in the machine scheduling literature. These include: Khosla I. (1995), Werner and Winkler (1995), Dauzere-Peres and Paulli (1997), Nowicki Allahverdi et al (1999), Daniels et al (1999), (1999), Steinhofel (1999), Mastrolilli and Gambardella (2000), Mati et al (2001), Mascis and Pacciarelli (2002), Kim et al (2003), Kis (2003), Corry and Kozan (2004), Murovec and Suhel (2004) and Zoghby et al (2005). In summary this literature addresses scheduling problems with routing flexibility, capacitated buffers, sequence dependant setup times, parallel machines, and more complex technical constraints.

A makespan objective criterion is used in this paper to measure the relative merits of a new timetable though other criterions could easily be used as our approaches are quite generic. The makespan objective is a well known scheduling measure and provides a good benchmark for comparing the efficiency of the techniques proposed in this paper. In our experience train scheduling criteria vary from one region and operator to the next and when constructing a new timetable the best objective criterion is particularly debatable. What is clear though is that new schedules are not affected by previous "timings". Therefore minimising delays such as those caused by the non-adherence to an existing schedule is not applicable. Furthermore minimising scheduled delays is not entirely sufficient because trains may be scheduled with no delays but the schedule horizon (makespan) can be very large. In other words throughput will be very poor and this is not particularly desirable. New timetables should be efficient in terms of throughput at least in certain time periods and the makespan objective is good for achieving this. The makespan minimisation criterion is also particularly useful as it allows the capacity of the system for a specific mix of trains to be accurately determined. No other fool proof method exists to our knowledge. In this scenario timetable creation may be viewed as a tool for making higher level economic decisions. For more information on capacity determination approaches and theory Kozan and Burdett (2005) and Burdett and Kozan (2006) may be consulted.

In the next section unique characteristics of train scheduling are first incorporated into the disjunctive graph representation of the solution. Constructive algorithms that utilise this representation are then developed in section 3. In section 4 the details of the meta-heuristics are presented. A numerical investigation and case study then demonstrates in section 5 the suitability of the proposed approaches and the quality of solution that can be obtained. In the last section the outcomes and the significance of the paper is summarised and the future research directions are given.

## 2. Model Development

Timetables are typically constructed manually or heuristically by manipulating the schedule representation of the solution. This is a list of train arrival and departure times at discrete identifiable locations (such as signalling device) along a railway line. The schedule may alternatively be represented by a list of entry and exit times for the sections of rail that occur between each pair of adjacent locations. In addition a sequence based representation of the problem is also possible. For example a schedule may be represented as a unique sequence of train movements on each section of the railway. The schedule and sequence are more or less equivalent because each can be obtained from the other. While one set of sequences can be obtained from each schedule, this is not necessarily true in the reverse sense. For example for the makespan objective train operations may be scheduled as early as possible, late as possible or anywhere in between.

When entry and exit times or sequences are used to represent a train timetable the problem is best described by the job shop scheduling framework and this is due to its generality and comprehensiveness. In a job shop approach for train scheduling, trains and sections respectively are synonymous with jobs and machines. **It should be especially noted that job and train and section and machine respectively are used interchangeably in the remainder of the paper.** The indices *i* and *j* are used to denote train (job) and section (machine) respectively. The set of jobs and machines is *J* and *M* respectively. Train scheduling is however not a classical variant of the job shop scheduling problem. An operation is now regarded as the movement / traversal of a job across a machine and not only as the processing of a job on a machine. An operation is denoted by $o_{i,k}$ and represents the traversal of train *i* across its *k*th section which is denoted by $m_{i,k}$. The time to traverse a section is $p_{i,k}$ and is also known as the sectional running time (SRT). The term "stage" is used to describe index *k* and the total number of stages is denoted by $K_i$. The list of machines visited by the train is the train route.

Train scheduling problems can not be modelled as classical job shops because of a number of features that are not commonly accommodated by the machine scheduling perspective but are common in railways. They are as follows:

- Trains and sections have length whereas jobs and machines do not. The length of train *i* and section *j* (in kilometres) is defined as $l_i$ and $l_j$ respectively (the index distinguishes which length is referred to).
- A train may be on multiple sections at one time.
- Each train operation does not take a pre defined amount of time, for example if acceleration and deceleration and variable velocities are included. The velocity of a train in particular is normally assumed to be fixed at some upper level, but realistically a train may travel at any speed below that.
- Once visited, a section may be revisited. For example, a train may reach a location and then returns to its starting point. In job shops the usual assumption is that jobs visit a machine once.
- After a job is processed on a machine, the next job may not begin immediately because the current jobs path is blocked or there is an imposed setup / separation time. The setup time of operation $o'$ if preceded by operation *o* (i.e. the finish-start headway (separation) between operation *o* and $o'$) is given by $s_{o,o'}$.

- Passing loops and other passing facilities are equivalent to parallel machines or capacitated buffers which are very difficult extensions of the standard job shop for which no efficient solution has yet been found.
- A non delay scheduling policy is usually assumed. That is, unforced idle time is not allowed and operations of a job should be processed immediately on a machine if that machine is ready. In other words operations must be scheduled as early or as late as possible, i.e. forwards or backwards scheduling is performed.

The differences above need to be incorporated in order for train scheduling to be performed realistically. In the remainder of this section these features are incorporated by changing and modifying the classical activity on node (AON) disjunctive graph structure of the job shop. For the standard JSP, nodes and arcs respectively represent operations and the precedence's between operations. Arcs are defined as either conjunctive or disjunctive and have a weight of zero. Disjunctive arcs in particular represent precedence's between operations of different jobs while conjunctive arcs represent precedence's between operations of the same job. The set of conjunctive and disjunctive arcs is denoted by $A$ and $E$ respectively and the set of operation nodes is set $V$. Nodes weights are equal to the operation processing time. A source node and a sink node are also added to the graph. The first and last operation of each job is attached from the source and to the sink node respectively. The longest path from the source to the sink node defines the schedule and gives the makespan. A new schedule may be obtained for example by selecting and reversing "critical" disjunctive arcs. Reversing a disjunctive arc is equivalent to reversing the position of two jobs within a machine sequence.

It should be noted that the non delay scheduling policy and fixed train speeds are retained in our approach. The non-delay scheduling policy is retained because allowing unforced idle time does not usually result in further improvements in the makespan. Removal of the non-delay policy is possible and could be incorporated at a later stage. Currently no mechanism other than a mathematical programming model exists to accomplish this. Fixing train speeds greatly reduces the complexity of the problem without significantly affecting the solution quality. In the context of constructing a new timetable and or determining the capacity of a system this assumption is more than reasonable and the reduction in the level of realism is minimal.

## 2.1. Train Length and Dwell Times

When the length of a train is neglected (i.e. a train is represented as a point) a section may be designated as being unoccupied when it isn't. This occurs when the rear of the train has yet to leave the section. Modifying the sectional running time (SRT) (i.e. the time to travel across the section) to include the time for the rear to exit the section however is insufficient as the actual entry time of the front will be incorrect (i.e. overestimated). The correct incorporation of train length results in overlapping operations in this hybrid job shop. A train operation may in fact begin before several predecessor operations are complete if the train is very long and or the previous sections are very small.

To incorporate train length the front of the train is explicitly modelled while the rear is not. For example the movement of the front of the train on each section is represented as a separate node in the disjunctive graph. The rear of the train is then modelled by making two alterations to the graph. The arc between the last node for the train (i.e. last train operation) and the sink node is given a weighting equal to the standard time lag (*stl*). The standard time lag is defined as the time for the train to traverse a distance equal to its own length while travelling at its regular / specified speed. This ensures that the train departs the system at the correct time. The standard time lag for train $i$ in minutes is $stl_i = 60\ l_i / v_i$ where $v_i$ is the

speed of train *i* in km/h. The standard time lag may be defined for each and every section that is traversed if the speed of the train is not constant.

The second alteration is to add the standard time lag to the disjunctive arcs which normally have a weight of zero. This then ensures the correct precedence relationship between the rear of one train and the front of another. This approach is simpler and more elegant than other competing approaches that were developed that are outside the scope of this paper. In particular it requires fewer nodes and fewer alterations to the original graph.

Trains may also be required to stop at certain locations for pre-specified periods of time. In this paper, dwell time denoted by $\delta_{i,k}$ is manifested at the section boundary in the direction of travel (i.e. a train stops just prior to leaving a section) and is incorporated as additional machine processing time. That is, dwell time is added to the node weight of the disjunctive graph. Dwell time however is defined separately from the machine processing time because of additional complexities that arise when a train's length exceeds the current section. This is because additional occupation time is required on the previous section over and above the planned sectional running time. Dwell times are not just incorporated as additional node weights; they are utilised as additional time lag. A graphical demonstration of time lags and additional time lags in a distance versus time line chart is shown in Figure 1.

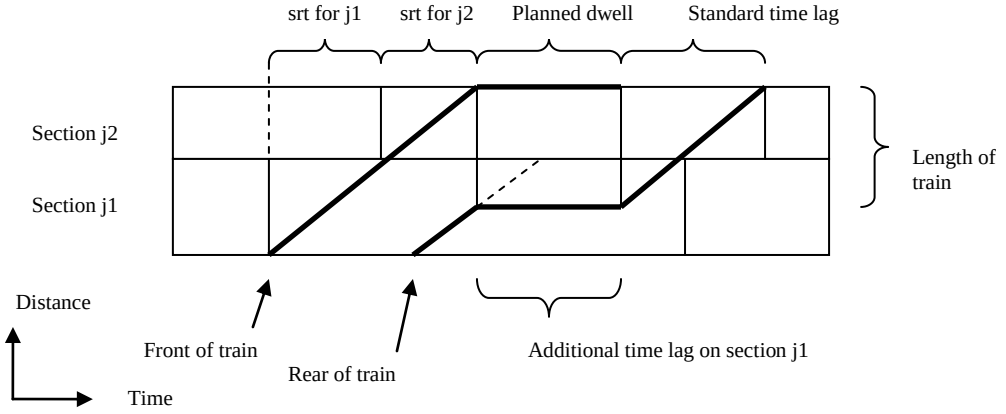



**Figure 1.** A graphical demonstration of time lags

For operation $o_{i,k}$ the additional time lag denoted by $atl_{i,k}$ is calculated by the following equation:

$$atl_{i,K_i} = 0\ ,\ atl_{i,k} = \sum_{k' \in \Im_{i,k}} \delta_{i,k'} \qquad \forall i \in J;\quad 1 \le k < K_i$$
$$\text{where } \Im_{i,k} = \ k' \mid k+1 \le k' \le K_i, dt_{i,k} > dt_{i,k'} - l_i \tag{1}$$

In the above equation the distance travelled up to (and including) the *k*th section is $dt_{i,k}$. In addition, $\Im_{i,k}$ is the set of all later train operation that affect $o_{i,k}$. These operations $o_{i,k'}$ affect operation $o_{i,k}$ because the elapsed distance $dt_{i,k'}$ is not sufficiently far away, i.e. $dt_{i,k'} < dt_{i,k} + l_i$. The elapsed distance can be calculated in the following way:

$$dt_{i,0} = 0, dt_{i,k} = dt_{i,k-1} + l_j \quad 1 \le k \le K_i, j = m_{i,k}, dt_{i,K_i+1} = dt_{i,K_i} + l_i \qquad \forall i \tag{2}$$

## 2.2 Headways (Separation)

In general train mass is very large and this causes poor stopping performance. Consequently for safety reasons it is not permitted for trains to be close to each other in case of collision. Therefore headways are utilised to facilitate the safety of all trains. Headways may be measured from the exit or entry time of one train and the entry time of another, and are called finish-start (F-S) or start-start (S-S) headways respectively. In this paper F-S headways are utilised. These values may be positive or negative depending on whether a section occupation condition of one train is required or not respectively. Headways are viewed as sequence dependent setup times and are incorporated in the graph representation as additional disjunctive arc weightings. These values are known (i.e. defined and or computed) beforehand. For more details on the calculation of headway parameters we refer the reader to Burdett and Kozan (2004) for example.

## 2.3 Blocking Conditions

Unlike the standard job shop, jobs do not automatically leave a machine after processing is completed. This is because after a train has traversed a section, its path may be blocked on the next section by another train. A section is deemed unoccupied only when the rear of the previous train has entered its next section. Therefore an operation performed on another machine must be inspected in order to establish occupancy about the current machine. **Consequently pairs of disjunctive arcs are not the reverse (as in classical job shops) and the sequence is not given by a continuous chain of disjunctive arcs; conjunctive arcs are also present.**

To enforce proper blocking conditions, **different** disjunctive arcs must be generated which are reliant upon additional "train specific" parameters. The parameters that must be computed are the stage and position of the front when the rear is departing the *k*th section in the route, denoted by $fstage_{i,k}$ and $fpos_{i,k}$ respectively. They are calculated in the following way.

$$fstage_{i,k} = \min\ k' \mid k+1 \le k' \le K_i + 1 \wedge dt_{i,k'} \ge dt_{i,k} + l_i \quad \forall i, k = 1,..,K_i \tag{3}$$

$$fpos_{i,k} = dt_{i,k} + l_i - dt_{i,\,fstage_{i,k}-1} \quad \forall i, k \tag{4}$$

The front of the train must be in the next adjacent section $m_{i,k+1}$ which is the next stage or else in another later section which is a later stage again. The actual position of the front lies between $dt_{i,\,fstage_{i,k}-1}$ and $dt_{i,\,fstage_{i,k}}$ as $dt_{i,k} + l_i \ge dt_{i,\,fstage_{i,k}-1}$ .

## 2.4. Train Speeds

In some train timetabling problems a constant train speed may be taken. For example the sectional running times on all section are proportional to this speed and remain static. There are several advantages to such an approach. For example the exit time of the back end can be accurately modelled because a constant speed causes a constant (and standard) time lag on each section. The time lag being the time for the rear to exit a section after the front has already exited.

However when train scheduling in practice it becomes apparent that sectional running time can be affected by track degradation, track curvature, track gradient and speed limits. Consequently the assumption of constant speed on every section can not be realised. The

sectional running times can not be simply modelled and hence actual sectional running times must be measured and not computed. If the first approach is used then small "incorrect" delays may be created. These delays are caused because of the usage of incorrect time lags when determining exit times. These delays can cause existing procedures to "fall over". What is proposed therefore is the selection of a speed on each section (and the computation of SRT's) or the computation of a speed based upon input sectional running times. ***The result is still a fixed speed model, but not a constant speed model.***

The time lag on each section must be computed and will no longer be constant. The time lag can not be simply computed as $lag_{i,k} = 60\ l_j / V_{i,k}$ where $j = m_{i,k}$ due to the effect of train length and dwell profiles on nearby sections. The new equations for computing the correct time lag can be expressed in the following way:

$$lag_{i,k} = \sum_{k'=k+1}^{fstage_{i,k}-1} srt_{i,k'} + \delta_{i,k'} + srt_{i,z}\ \ fpos_{i,k} / l_{m_{i,z}} \qquad \forall i,k \tag{5}$$

where $z = K_i$ if $fstage_{i,k} > K_i$ and $z = fstage_{i,k}$ otherwise. The planned occupancy time on each section is $ot_{i,k} = srt_{i,k} + \delta_{i,k} + lag_{i,k}$. The additional time lag and standard time lag parameters used in previous sections become redundant as a consequence of using this more accurate time lag calculation.

## 2.5 Passing Loops

Passing loops (also known as crossing loops) allow trains to overtake or pass each other at predetermined positions and consist of at least two parallel tracks or sometimes more. Without passing facility, only uni-directional flow is possible on a single track. Passing loop incorporation is an important and complex facet of the train scheduling problem. In theory passing loops may be modelled explicitly or implicitly. An explicit approach represents each track as an additional "sequence-able" machine. This causes additional routing alternatives (routing flexibility). An implicit approach represents the passing loop as a capacitated intermediate storage area (buffer). The buffer is itself modelled as an additional machine with no sequence and no associated disjunctive arcs. It should be noted that buffer machines must also be explicitly included in a jobs path.

The buffer approach assumes that passing loop length and processing time is fixed. However in reality, each track may be a different length and may take a different amount of time to traverse for example due to safety conditions and track curvature. Consequently the routing flexibility approach for the representation of passing loops is more accurate than the buffer approach. Another advantage of the routing flexibility approach is that the contents of the passing loop are explicitly known, i.e. there is a sequence for each track. The main advantage of the second approach is that there are far fewer machines in the problem. There are also no alternative routing complexities and consequently the disjunctive arc data remains static. However buffer occupancy violations (BOV) may occur and must be identified and resolved.

In this paper either approach or a combination of both may be taken. The first approach however is required when there is legitimate routing flexibility that can not be modelled by capacitated buffers. Buffer occupancy may be determined after the disjunctive graph has been evaluated and decoded. In particular the operation entry and exit times may be used as they provide an interval of machine occupancy. The original overlapping intervals of machine occupancy are split into separate non-overlapping intervals, each with its own integer

parameter signifying the number of occupants. In other words a machine occupancy record (MOR) is a three tuple $start, end, number$ which signifies the interval of occupancy and the number of job occupants. If the number of occupants of any non-overlapping interval exceeds the capacity of the buffer, then a buffer occupancy violation (BOV) has been found.

The complete ordered list of non-overlapping intervals is required in order to prove that a capacitated buffer is not overloaded. The process of finding all the non overlapping intervals is achieved by an efficient iterative algorithm. First the original intervals of machine occupancy are ordered chronologically. Adjacent pairs of intervals (starting from the first) are then sequentially compared. For example two intervals [*a*,*b*] and [*c*,*d*] are deemed to overlap if *b* is greater than *c*. The two intervals may be split in the following ways: ( [*a*, *b*] ), ( [*a*, *b*], [*b*, *d*] ), ( [*a*, *c*], [*c*, *d*], [*d*, *b*] ), ( [*a*, *c*], [*c*, *d*] ) and ( [*a*, *c*], [*c*, *b*], [*b*, *d*] ). When two intervals overlap, zero to three new sub intervals may be created. However at most, only one must be inserted into the set according to the ordering condition. The modification of the original intervals ensures that the unnecessary creation of new intervals and their subsequent insertion is not performed. It should also be noted that if the newly created interval already exists then the current occupancy level is increased and the interval is not re-inserted. Due to an interval ordering condition no sub interval can be inserted before position *k*-1 and hence the *k*-1st MOR is never removed, although it may be modified. This means that the comparison process can continue without starting from the beginning when the set of intervals is enlarged or reduced.

## 2.6. The Disjunctive Graph Model

In this section the complete disjunctive graph model is presented as the necessary elements needed to construct it have all been discussed. It should be noted that this new disjunctive graph model operates in the same fashion as explained for the classical JSP. Its structure however is different.

$$V = \{so, 0.0\} \cup \bigcup_{\forall i \in J} \bigcup_{k=1}^{K_i} \{o_{i,k}, p_{i,k} + \delta_{i,k}\} \cup \{si, 0.0\} \tag{6}$$

$$A = \bigcup_{\forall i \in J} \{so, o_{i,1}, rlt_i\} \cup \bigcup_{\forall i \in J} \bigcup_{k=2}^{K_i} \{o_{i,k-1}, o_{i,k}, 0.0\} \cup \bigcup_{\forall i \in J} \{o_{i,K_i}, si, stl_i\} \tag{7}$$

$$E = \bigcup_{j \in M} \bigcup_{\forall \{o_{i,k}, o_{i',k'}\} \in E_j} e_1 \cup e_2 \tag{8}$$

$$e_1 = \begin{cases} \{o_{i,k^*}, o_{i',k'}, s_{o_{i,k}, o_{i',k'}} - \delta_{i,k^*} - p_{i,k^*}\} \; 1 - fpos_{i,k} / l_{m_{i,k^*}} & \text{for } k^* = fstage_{i,k} \le K_i \\ \{o_{i,k^*}, o_{i',k'}, s_{o_{i,k}, o_{i',k'}} + p_{i,k^*}\} \; fpos_{i,k} / l_{m_{i,k^*}} & \text{for } k^* = fstage_{i,k} > K_i \end{cases} \tag{9}$$

$$e_2 = \begin{cases} \{o_{i',k^*}, o_{i,k}, s_{o_{i',k'}, o_{i,k}}, - \delta_{i',k^*} - p_{i',k^*}\} \; 1 - fpos_{i',k'} / l_{m_{i',k^*}} & \text{for } k^* = fstage_{i',k'} \le K_{i'} \\ \{o_{i',k^*}, o_{i,k}, s_{o_{i',k'}, o_{i,k}} + p_{i',k^*}\} \; fpos_{i',k'} / l_{m_{i',k^*}} & \text{for } k^* = fstage_{i',k'} > K_{i'} \end{cases} \tag{10}$$

According to (6) each train operation has an associated node in the graph. The conjunctive arc set is constructed in (7); each three tuple represents an arc between two nodes with the given arc weight. The release time of train *i* and train operation $o_{i,k}$ is denoted by $rlt_i$ and $rlt_{i,k}$ respectively. Release times are incorporated for additional timing constraints that may be

imposed by planners. It should be noted that whenever a trains route is modified (as in the routing flexibility approach for modelling passing loops) set *A* must also be modified. The set of disjunctive arcs is created by expression (8)-(10). Equation (8) states that there are disjunctive arcs associated with every section (i.e. non buffer machine). On each of these sections and for each pair of train operations $o_{i,k}, o_{i',k'}$ that use this section there is a pair of arcs as defined by $e_1$ and $e_2$, one for the precedence $o_{i,k} \prec o_{i',k'}$ and one for the precedence $o_{i',k'} \prec o_{i,k}$ respectively. Equations (9) and (10) are based upon the logic provided in sections 2.1 – 2.3. As this problem is not a classical job shop the disjunctive arcs are not $o_{i,k}, o_{i',k'}$ and $o_{i',k'}, o_{i,k}$ . The length of trains is included and the arcs must start from other operation nodes namely $o_{i,k^*}$ and $o_{i',k^*}$ which are later operations of train *i* and $i'$ respectively. The arc weights in essence project backwards from these later nodes and take into account the processing requirements that occur between $o_{i,k}, o_{i,k^*}$ and $o_{i',k'}, o_{i',k^*}$ respectively (where $k^*$ is different in each).

To decode a particular disjunctive graph after the longest path algorithm has been applied (i.e. to actually schedule train operations) the following equations are necessary:

$$entry_{i,k} := lpv_{i,k} - p_{i,k} - \delta_{i,k} \qquad \forall i \in J, k = K_i,..,1 \tag{11}$$

$$exit_{i,k} := lpv_{i,k} + stl_i \qquad \forall i \in J, k = K_i \tag{12}$$

$$exit_{i,k} := lpv_{i,k^*} - \delta_{i,k^*} + p_{i,k^*}\left(1 - fpos_{i,k} \middle/ l_{m_{i,k^*}}\right)$$
$$\forall i \in J, k = K_i - 1,..,1 \mid k^* \le K_i \text{ where } k^* = fstage_{i,k} \tag{13}$$

$$exit_{i,k} := lpv_{i,k^*-1} + p_{i,k^*-1}\left(1 - fpos_{i,k} \middle/ l_{m_{i,k^*-1}}\right)$$
$$\forall i \in J, k = K_i - 1,..,1 \mid k^* = K_i + 1 \text{ where } k^* = fstage_{i,k} \tag{14}$$

$$delay_{i,k} = exit_{i,k} - entry_{i,k} + p_{i,k} + \delta_{i,k} + lag_{i,k} \qquad \forall i \in J, k = K_i,..,1 \tag{15}$$

In these equations $entry_{i,k}$, $exit_{i,k}$ and $delay_{i,k}$ are the scheduled entry, exit time and delay for train operation $o_{i,k}$ . The value of the longest path to operation $o_{i,k}$ is $lpv_{i,k}$ . The longest path to a particular node represents the completion time of the front but not the departure/exit time for the rear. The entry time is simply the completion time of the front minus any section occupation time (SOT) that it must incur. There are three cases for determining the correct exit time for an operation. Determining the exit time is more complex because the rear of the train must be taken into account. Equation (12) is valid for the last stage where the front and rear depart straight away. Equation (13) and (14) are valid respectively for all stages (except the last), where the front still lies within the system or where the front has departed the system. In these equations the exit time of the rear is obtained by projecting backwards and forwards respectively from the entry time of the front on another section.

The disjunctive graph may alternatively be evaluated in a backwards fashion so that operations are scheduled as late as possible as opposed to as early as possible. For this backwards scheduling option, the longest path values must be converted to forwards time. This is achieved by subtracting from the makespan the current longest path value and then adding the node weight.

## 3. Constructing a Schedule

A constructive algorithm is proposed for sequencing trains as a standalone approach or to provide starting solutions for meta-heuristic strategies. There are two variants of the constructive algorithm. The first is primarily for the situation where passing loops are modelled as capacitated buffers and train routes are fixed. The second variant is for the other situation where passing loops are modelled as additional machines and additional routing decisions must be made. In other words jobs have selectable and not fixed routes.

To determine a feasible schedule it is proposed that trains are inserted iteratively (i.e. one by one). The operations of each train are also inserted iteratively. The order that trains are inserted is from largest to smallest total transit time (processing time) though a number of other alternatives could be taken. The way in which trains are inserted depends on the status of certain user defined boolean (binary) flags. The flags are *re_route*, *route_greedy*, *insert_greedy* and *permit_BOV* and their affect should become apparent as the details of the constructive algorithm are further discussed. The methodology of the constructive algorithm is very much the same as that of the NEH insertion algorithm of Nawaz et al (1983) for the flow shop in which the sequence is constructed one job at a time. The main difference is that there are *m* sequences instead of one and each operation of a train must be separately inserted. Further details are as follows:

### 3.1. Fixed Route Variant

The ***InsAlg_1*** procedure attempts to insert trains with fixed routes. The operations may be inserted in a forwards or backwards manner with respect to the route, i.e. from first to last or last to first operations. The backwards approach however has been taken in order to remain compatible with other procedures. For each operation performed on a standard machine (i.e. for non buffer operations), a test insertion phase is performed. Each position in the current partial sequence is inspected for feasibility by temporarily inserting the operation and then re-evaluating the disjunctive graph. The insertion of an operation causes one or two disjunctive arcs to be added to the graph. An existing disjunctive arc is made redundant if the operation is not inserted in the first and last position of the current partial sequence. It is not required for redundant arcs to be removed in this procedure however. Redundant arc are only removed when the insertion position is finalised and the operation is inserted permanently.

If a cycle occurs in the graph, the position is marked as infeasible. If the position is feasible and the *permit_BOV* flag is false then buffer machine occupancy is determined. If no violations have occurred and the insertion results in the smallest makespan, it is recorded as the best insertion point found so far. The operation is then removed and the graph is returned to its previous state. At the end of this phase the insertion of the operation is made if a feasible insertion point exists. For the *insert_greedy* equals true case (i.e. the greedy local search option) the operation is inserted in the best position, otherwise a position is chosen randomly. The insertion point is then flagged as infeasible in case backtracking is required at a later stage. The operation counter is then decremented and the next operation is tested for insertion in the same manner. Buffer operations are automatically inserted but are not sequenced.

It should be noted that a very important part of the test insertion phase that is associated with re-entrant paths was not mentioned above. For example, a train operation ***can not*** be inserted before predecessor operations of the same train that traverse the same section because the precedence conditions would be violated. Therefore insertion positions before the last occurrence of the train on the section are not inspected. To facilitate this, a variable that stores the last insertion position is created and continually inspected and updated whenever necessary.

If at any stage an operation may not be feasibly inserted into the partial sequence for the associated section, a backtracking phase is initiated. Firstly the operation counter is incremented. If the value is greater than $K_i$ the algorithm is terminated because no further backtracking is possible, and the train is not inserted. Otherwise the insertion of the previous operation is undone, i.e. the operation is removed and the graph is returned to a previous state. The previous operation re-enters the test insertion phase but the previous insertion point is not re-inspected as it has been flagged as infeasible.

### 3.2. Flexible Route Variant

The ***InsAlg_2*** procedure attempts to insert trains that have flexible routes. It is similar but more complex than the ***InsAlg_1*** algorithm. The main conceptual difference is that a number of alternative operations may be selected at each stage of the algorithm and subsequently added to a trains route. In other words the route is built at the same time as the train is inserted (scheduled). The procedure starts from the sink node in the train's precedence network and continues until the source node is reached. For each alternative operation each possible insertion position is evaluated and compared. For the *route_greedy* case the operation with the best insertion position is chosen, otherwise an operation is chosen randomly. It should be noted that for standard train scheduling problems the computational burden will not be significantly greater since the number of alternatives will be small, i.e. two for a standard passing loop. The backwards approach must be taken here because the route is not known and because of the blocking conditions for an operation that require information about a successor operation. For example the successor operation is not known when the route is constructed in a forwards manner and hence an operation can not be inserted correctly.

There are several sources of additional complexity. The first is the difficulty in storing and maintaining backtracking information. For example, when the routes are fixed, the dimensions of the backtracking parameters are also fixed. When the routes are constructed dynamically, the dimensions are constantly increasing and decreasing. There is also more backtracking information "floating about" due to the choice of additional operations at each stage. The second source of additional complexity is associated with the re-calculation of train parameters and disjunctive arc information. For example all disjunctive arc information may be preliminary calculated when train routes are fixed. However disjunctive arc information must be continually updated when train routes are dynamically constructed. It should be noted that it is not computationally reasonable to preliminarily compute disjunctive arc information for all possible routing possibilities and precedence relationships between trains. The parameters associated with the position of the front in particular also changes and must be updated when the route changes.

## 4. Meta-Heuristic Improvement

In recent years meta-heuristics have been applied greatly to job shop scheduling problems particularly classical. Examples include Van Laarhoven et al (1992), Steinhofel et al (1999), Kolonko (1999), Nowicki (1999), Mastrolilli and Gambardella (2000), Murovec and Suhel (2004), Corry and Kozan (2004), Zoghby et al (2005), Groflin and Klinkert (2007). Meta–heuristic approaches however must be adapted and extended for train scheduling problem.

The application of the Simulated Annealing and Local Search meta-heuristic are concentrated upon in this paper (and section). Tabu Search (TS) was not investigated because the possibility of asymmetric neighbourhood stops cycling (Kolonko 1999). Evolutionary Algorithms (EA) including Genetic Algorithms (GA) were not investigated because they are

expected to be inferior on this type of discrete problem. This assumption is based upon past experience. In particular the crossover mechanism will in all likelihood result in vastly infeasible solutions that will be difficult if not costly to repair. The offspring solutions will also more than likely be completely different from their parent solutions and will be of inferior quality. Lastly the computational burden (in terms of CPU time) of manipulating a population of solutions is too large. Several EA and GA approaches however have been applied to some classical and non classical job shop problems in the past. Most recent examples are Mattfeld and Bierwirth (2004), Kim et al (2003) and Candido et al (1998). None of these papers however compare their approach with other meta-heuristics.

### 4.1. BOV Handling

The BOV's that are created in the course of perturbing a solution may be dealt with in one of three ways. They may be allowed but penalised in the objective function. Secondly they may not be allowed at all, or thirdly they may be resolved. Resolution of buffer occupancy violations explicitly is expected to be a computationally time consuming task and the code and algorithms required more complex. Consequently this approach is not taken. It should be noted that if the starting solution contains BOV's (i.e. it is infeasible) then the penalisation option must be used otherwise a feasible solution can not be obtained. If the starting solution contains no BOV's (i.e. it is feasible) then either the penalisation or restrict option may be used.

The main disadvantage with penalising the objective function is that a feasible solution may not be obtained nor is it guaranteed. However this approach may allow better solutions to be reached that would not otherwise be reachable (i.e. by the restricted case). There are several ways in which to penalise the objective function and two alternatives are shown:

$$OBJV = Cmax + \rho \times \left| BOV \right| \text{ or } OBJV = Cmax + \rho \sum_{k \in BOV} \upsilon_k - cap_{j_k} \tag{16}$$

In these equations $cap_j$ gives the capacity of buffer machine *j*, $U_j$ is the set of operations that require machine *j*, and the set of buffer occupancy violations is $BOV = \{ j,\upsilon \mid j \in M, 1 \le \upsilon \le |U_j| \}$ which is a set of 2-tuples. Each element signifies the associated machine and the number of occupants respectively. The first equation only penalises the occurrence of a violation while the second also penalises the extent of the violation. The penalty value $\rho$ may be fixed or it may be variable. A variable penalty value could be increased as the temperature in a Simulated Annealing approach is reduced. Similarly a variable penalty value could be increased as the number of iterations in a Tabu or Local Search approach is increased.

The first approach for penalising BOV's was taken. The second approach was not taken because no great benefit was observed. In particular buffers are seldom overloaded by more than a few jobs and the extent of the difference is small. If the penalty value is large then overloading the buffers is not acceptable as a consequence of a move. If the value is small then it is very difficult to improve the objective function and maintain correct buffer levels. Furthermore resolving heavily overloaded buffers causes unreasonable amounts of congestion which again leads to inferior solutions.

### 4.2. Simulated Annealing Details

The standard SA control structure was used. For example there is an outer temperature loop which is terminated when the number of temperature steps reaches a predefined limit and an inner loop which is terminated after a given number of state changes at the given temperature. A state change is accepted using the standard metropolis function. Lastly the temperature is altered at the end of the inner loop. The basic geometric cooling schedule was used and the initial temperature was chosen using past experience and experimentation.

A new feature that is added to the SA approach is the addition of the BOV penalty value. The penalty value $\rho_z$ is increased proportionally as the temperature $t_z$ decreases. If the initial penalty value is $\rho_0 = P_I$ and the final penalty value is $P_F$ then an expression for the penalty value at step $z$ is $\rho_z = \alpha + \beta\ \gamma^{-z}$ where $\alpha = \frac{P_I\ \gamma^{-Z}\ - P_F}{\gamma^{-Z}\ -1}$ and $\beta = \frac{P_F - P_I}{\gamma^{-Z}\ -1}$. This equation was derived by firstly observing that $\gamma^{-z}$ gives the correct relationship between the number of steps and the increasing penalty value. The correct values of $\alpha$ and $\beta$ are obtained by substitution using $\rho_0 = P_I$ and $\rho_Z = P_F$.

An important feature of the changing penalty value that is not immediately visible is the effect on the best solution and the current solution. When the penalty value is altered at the end of the inner loop after the temperature has been altered, the best and current solutions will change if they contain BOV's. Consequently a solution that is viewed as superior at one stage in the search may be viewed as inferior at a later stage. Consequently an additional procedure must be called to update these values if the search is to proceed correctly.

#### 4.2.1 Unitary Perturbation Operators

The solution which is a collection of operation sequences (i.e. one for each non buffer machine) may only be realistically and efficiently perturbed in a small number of ways. For example operations may be individually shifted, pairs of operations may be interchanged or exchanged, or a sub sequence of operations may be reversed. Entire jobs may also be removed and re-inserted using the constructive algorithm as an alternative perturbation strategy.

Reversal of a sub sequence is a poor strategy for train scheduling problems due to blocking conditions and the capacitated buffers. In particular if the buffers may not be overfilled then the reversal of most sub sequences will not be accepted as they will result in infeasible solutions. If buffer overflows are penalised, high levels of infeasibility will still likely be caused and those moves will be rejected unless the penalty value is very small. Consequently much of the computational effort will be wasted. For similar reasons, an exchange operator and a general shift move is also quite poor. Similarly the computational overhead of re-moving and re-inserting an entire job is much too great. The interchange operation which is equivalent to a shift move of one adjacent position (i.e. to the left or right) is therefore the most suitable "unitary" perturbation move.

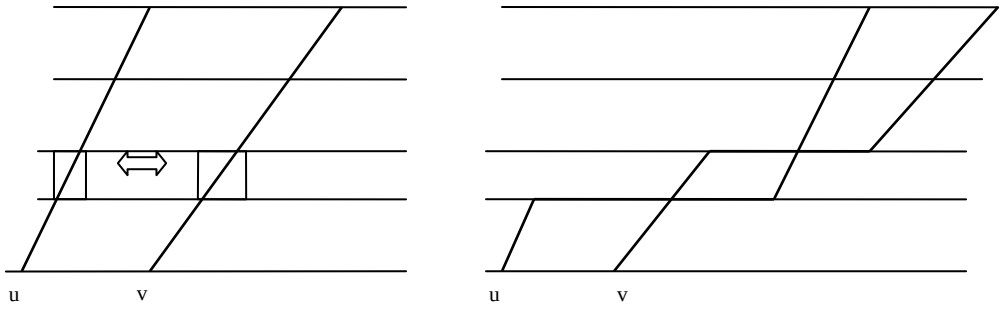
Three alternative operation selection strategies were implemented and investigated. The first is a random selection strategy. The second approach chooses operations based upon the level of delay that has been incurred. A roulette wheel selection is used and those operations that have been delayed are the most more likely to be chosen. The third strategy selects operations from the critical path. In each of the three strategies the set of selectable operations does not include "buffer" operations. This is because buffer operations are not sequenced

Apart from the different ways of selecting an operation the steps of the perturbation operator remain the same. If the position of the operation is first or last then there is only one possible shift that may be taken; right and left respectively. Otherwise the shift direction is chosen randomly. **An operation however must not be shifted past any predecessor or successor operations of the same job on the same machine (OSM) otherwise errors will occur.** This occurs for example when trains revisit sections multiple times. Return paths are the most typical example in train scheduling problems.

It should be noted that when the critical path is used to select operations, the local search neighbourhood becomes asymmetric. Asymmetry implies that a solution y may be reached from x (i.e. $x \in N\ y$ ) but x is not necessarily reachable from y (i.e. $y \notin N\ x$ . A symmetric local search operator is normally assumed and utilised by meta-heuristics. The critical path is also a little different to that of the classical job shop and this is due to the blocking conditions, the capacitated buffers and re-entrancy issues. Typically the critical path is a list of operations linked by disjunctive arcs. The critical path in this paper now consists of both conjunctive and disjunctive arcs and has a "zig-zag" shape. Operations on the critical path may also be processed on buffer machines for which no sequence exists and no disjunctive arcs. Direct links between operations of different jobs no longer occurs except at machines where blocking conditions are not enforced (input-output points to be precise). Links between operations of different jobs occur through adjacent operations on adjacent machines.

### 4.2.2. Compound Perturbation Operators

From numerical investigations it was found that shifting an operation within a sequence is sufficient as a means of perturbing (refining) an existing train schedule and allows relatively good solutions to be obtained with reasonable computational expense (overheads). However it has also been observed that these moves often exhibit serious limitations because of the more complex and constrained search space for this type of problem. This is even more apparent on larger instances. For example the operators become less efficient as the number of trains and sections and hence the number of operations increases. To reach a better solution it often requires a large number of non improving moves to be made. What these non-improving moves are is not known. It is also quite difficult if not impossible to move from one feasible solution to another feasible solution at times without allowing BOV. Therefore it is quite clear that compound moves are necessary in order to find better solutions and in a more computationally efficient manner. It should also be noted that without the benefit of a compound move a single interchange may result in a multiple overtaking situation (conflict) as shown in Figure 2(b) which is highly undesirable due to the added congestion that is caused and the pointlessness of the scenario. In Figure 2 the diagonal lines represent train trajectories for two trains *u* and *v* respectively in a time versus distance chart that is commonly used to visualise train schedules. Stationary trains are identified by the horizontal lines in Figure 2(b). The rectangles in Figure 2(a) identify two train operations that occur on the same section of rail that for arguments sake are to be reversed in the associated machine sequence. The rectangles also demonstrate the section boundaries. The other train operations and are not explicitly shown nor are other trains that could be in the current schedule.



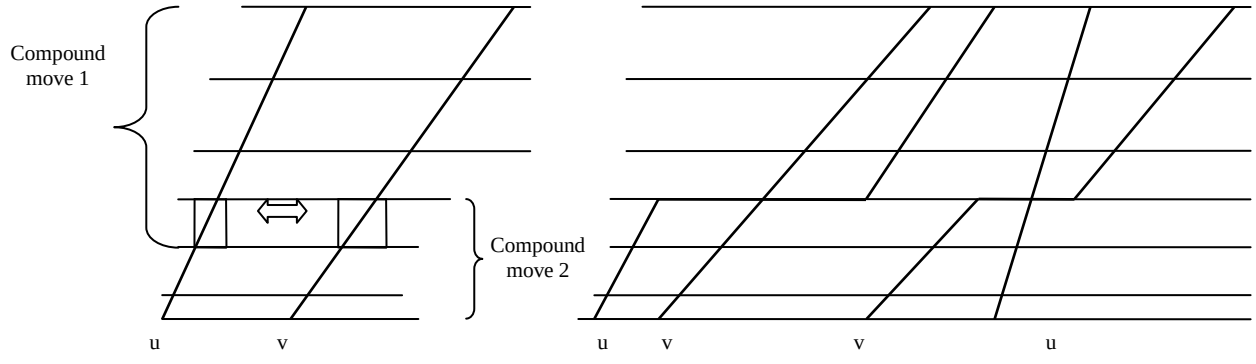

**(a)** **(b)**

**Figure 2.** Train chart demonstrating the creation of a multiple overtaking conflict

The focus of the proposed compound moves is to perform interchanges without causing multiple overtaking conflicts and secondly to move whole trains past each other easily within an existing schedule. In our approach simple interchanges and exchanges are used as the basis of the compound moves. An interchange swaps the position of two adjacent operations while exchanges swap the relative position of two operations that are not necessarily adjacent. The compound moves require several of these "single" moves to be performed "simultaneously".

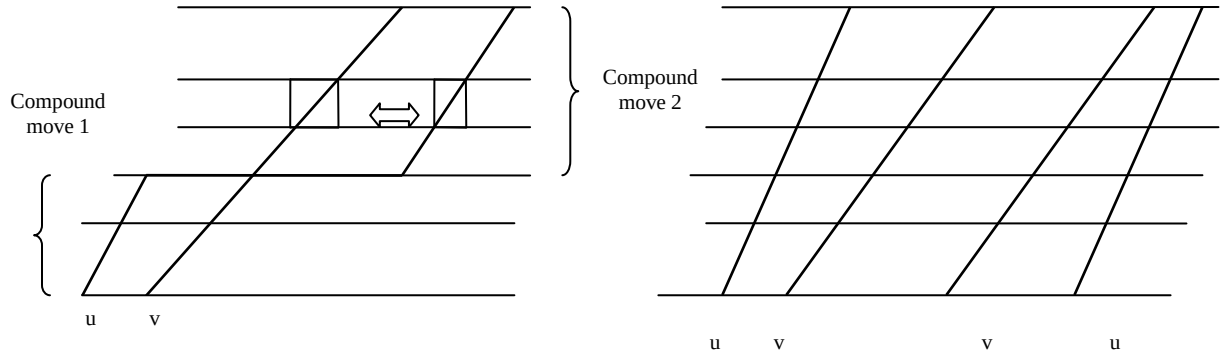
The compound moves are created by firstly choosing two operations. These operations must be adjacent in a machine sequence and must not be part of the same job. Each compound move therefore consists of at least one interchange move and the focus of the compound move occurs at this point (i.e. the machine). Each operation has a direction of travel (i.e. up or down) and therefore four cases can occur.

The first two cases result in a multiple overtaking situation as the trains both travel in the same direction. To stop this from occurring the order of the trains must be completely reversed or a single overtake can be allowed. The "make overtake" and "undo overtake" compound moves are shown graphically in Figure 3 and 4 respectively.




**(a)** **(b)** **(c)**

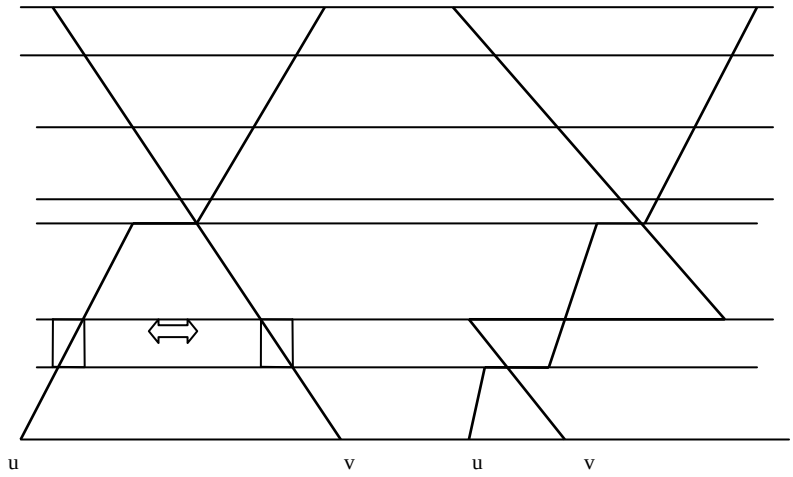
**Figure 3.** Train chart demonstrating a make overtake compound move




**(a)** **(b)** **(c)**

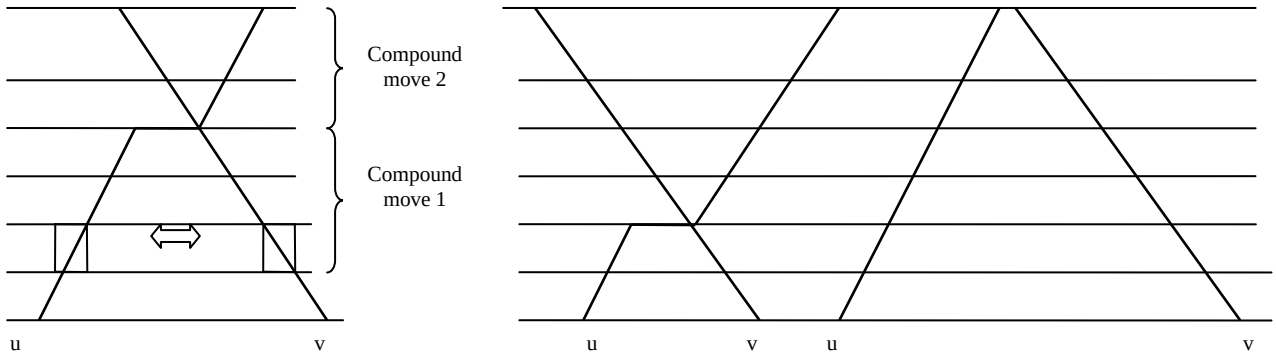
**Figure 4.** Train chart demonstrating an undo overtake compound move

When trains travel in opposite directions compound moves are also necessary because precedence impossibilities can result. An example is shown in Figure 5.



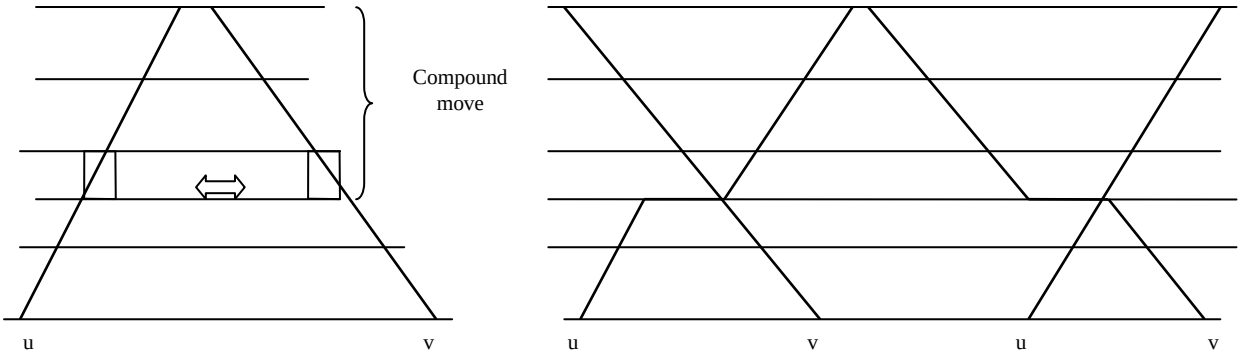

**(a)** **(b)**

**Figure 5.** Train chart demonstrating the creation of a precedence impossibility




**(a)** **(b)** **(c)**

**Figure 6.** Train chart demonstrating shift & undo passing compound moves




**(a)** **(b)** **(c)**

**Figure 7.** Train chart demonstrating make passing compound moves

Precedence impossibilities are identified during the evaluation of the disjunctive graph and when found the moves that cause them are usually reversed. It is better however not to make moves that cause precedence impossibilities if possible before the disjunctive graph is evaluated in order to reduce unnecessary "wasteful" calculations. Two compound moves can be applied for this scenario and are shown in Figure 6 and 7.

Trains may also be completely swapped as a compound move. This alternative though is not used because the schedule could be disrupted too greatly. If two trains do not interact with others then such a move is fine.

A compound interchange is formally defined as a three tuple $CPDINT = \{i, i', \Theta\}$ where $i \neq i'$ and $\Theta = \{o_{i,k}, o_{i',k'} \mid k \neq k', m_{i,k} = m_{i',k'}\}$. The size of the set of operation pairs $\Theta$ can not exceed the number of operations in both jobs, i.e. $|\Theta| \leq \min\{K_i, K_{i'}\}$.

Algorithm 1 (whose finer details can be found in the Appendix) coordinates the creation of compound moves between two operations. The set of machines common to each pair of jobs is first required. Two trains $i$ and $i'$ are deemed to travel in the same direction if they have the same route through the common machines. That is:

$$same\_dir = \bigcap_{z=1,\ldots,|common_{i,i'}|} m_{order_{i,z}} = m_{order_{i',z}} \qquad (17)$$

where $order_i$ and $order_{i'}$ are the order in which train $i$ and $i'$ traverse the common machines.

The position of the selected operations within the orderings is required next as this position is the "focal point" of the compound move. Following this the specific passing scenario is identified and sub procedures are called to generate the compound move. When two trains pass each other a compound move to shift the passing point or completely remove it are available. Which alternative is best is not known. The shift passing option is chosen at this stage because it encompasses the other strategy. In particular if the operation associated with the first common machine is selected then the passing is completely removed. In all other circumstances the passing is maintained but the position is shifted. The term passing is also used to signify overtaking. To identify passing Algorithm 2 (in the Appendix) compares the precedence's that occur between two operations on adjacent non buffer machines. Passing is identified when these precedence's differ.

The details of the ***MakePassing***, ***UndoPassing*** and ***ShiftPassing*** procedures are shown in Algorithm 3 – 5 of the Appendix. The main difference between these algorithms is the starting and finishing position for the compound move.

### 4.3. Local Search Details

A standard local search algorithm is used in this paper. For example there is a single loop which terminates after a specified number of steps (iterations) $Z$ have been performed. At each step all solutions in the current neighbourhood are evaluated and the best is chosen as the new current solution. This approach differs from Tabu Search in that there exists no tabu list mechanism (i.e. it does utilise memory) and no intensification and diversification strategies. This approach could be adapted to a Tabu Search approach but this is left for a future occasion. This technique does not have a temperature parameter and hence the penalty value must be modified according to the iteration counter. The following equation $\rho_z = P_I + \left( \frac{P_F - P_I}{Z^\varphi} \right) z^\varphi$ is used in this paper. This equation does not use the existing temperature reduction parameter but another parameter $\varphi$. Parameter $\varphi$ takes positive values greater than or equal to one. At one the relationship is linear. Larger values ensure that the increase in the penalty value is slower but more drastic towards the end of the search. As with SA, the current and best solutions should be altered to reflect the current penalty value. The penalty value is also changed at the end of each step after the neighbourhood has been searched.

As with SA a solution is again perturbed using shift moves. However in LS more than one move is evaluated at each step. The neighbourhood may be defined in many different ways. The following are three strategies that have been implemented and were investigated in this paper.

- **ShiftAll_Operation**: This involves the re-insertion of a single chosen operation and is equivalent to testing all possible shifts. The operation is chosen randomly or from the critical path. An operation may be re-inserted in one of the $|U_j|+1$ positions where $U_j$ is the set of operations that require machine *j*.
- **ShiftAll_Job**: This involves shifting the position of all operations (separately) of a chosen job by one position forwards (+1) or backwards (-1). The size of the neighbourhood for job *i* is twice the number of operations in its itinerary (route).
- **ShiftAll_Machine**: This involves shifting the position of all operations (separately) on a selected machine by one position forwards (+1) or backwards (-1). The size of the neighbourhood for machine *j* is $|U_j|-1$. Redundant moves are avoided by evaluating forward moves only. For example for two adjacent operations "*u*" and "*v*" shifting "*u*" forwards is equivalent to shifting "*v*" backwards. This operator is also equivalent to performing all interchanges of operations.

A fourth strategy in which all critical (non buffer) operations are (separately) shifted one position could also be taken and may be quite effective. This approach was not implemented as the former strategies were deemed to be more promising and secondly due to the different critical path property of this type of scheduling problem. For example, shifting only critical operations may not be sufficient to reach the optimal.

Each of the three strategies above must take into account the possibility of revisited machines like the perturbation operators of SA.

## 5. Numerical Investigation

### 5.1. Test Problems

The primary purpose of this numerical investigation was to identify the best solution approach for creating train timetables. More specifically the best meta-heuristic approach, perturbation operator, control parameters and BOV handling option are sought. The proposed techniques were judged according to the usual measures of performance CPU time and solution quality for a variety of interesting and increasingly demanding train scheduling problems. The results were obtained using a Pentium 4 3 Ghz computer.

A variety of solutions obtained from the constructive algorithm (and associated numerical investigations) were used as starting solutions for the meta-heuristics. The different SA and LS perturbation operators were also investigated. For SA the following control parameters were used.

10 / 0.01 / 0.95 / 50 → 134 / 6700 1.0 / 0.01 / 0.99 / 50 → 458 / 22900
10/ 0.01 / 0.99 / 50 → 687 / 34350 1.0 / 0.01 / 0.99 / 100 → 458 / 45800

From left to right these are: the initial temperature, the final temperature, the temperature reduction factor and the evaluations at each temperature step. From these values the total number of temperature steps and evaluations can be computed. Past experience and the results of preliminary experimentations dictated the choice of these particular parameters. The main LS parameter required is the number of iterations and three values 1000, 2000 and 3000 were investigated. The value $\varphi$=1 was chosen for increasing the penalty value when the penalisation option is selected. In total nearly 3000 test problem instances were solved.

Scheduling may be performed in a forward or backward manner and both were investigated because each utilises buffers differently. In each problem equal mixtures of 60, 80, 100, 120 and 160 km/h trains respectively were scheduled to make the problems more challenging. The variation in train speeds ensures that fast trains will run into slower trains quite easily and careful planning of passing and overtaking is necessary in order to maximise throughput. The test problem dimensions are summarised in Table 1. The problem sizes are quite typical of job shop scheduling instances found in recent papers and some smaller railway applications. It should however be noted that the added complexities of this problem (such as blocking, capacitated buffers, etc) make it more difficult to solve than classical job shop scheduling problems. This has been observed for example by Mascis and Pacciarelli (2002). These sized problems allow the techniques to be accurately judged with a reasonable amount of effort.

**Table 1.** Test problem dimensions

| Case | Type | Input / Output Points | Machines (Sections / Passing Loops) | Trains (Jobs) | Number of Operations |
|---|---|---|---|---|---|
| 1 | Serial (121.67 km) | 2 | 39 (20 / 19) | 20 | 780 |
| 2 | Serial (88.96 km) | 2 | 33 (12, 11) | 20 | 460 |
| 3 | Serial (260.25 km) | 2 | 45 (23, 22) | 20 | 900 |
| 4 | Serial (56.2 km) | 4 | 17 (10, 7) | 24 | 240 |
| 5 | Circular (39 km) | 1 | 8 (5, 3) | 12 | 60 |
| 6 | Network (188.6 km) | 5 | 41 (30, 11) | 54 | 1620 |
| 7 | Serial (467.76 km ) | 2 | 107 (55,52) | 30 | 3210 |

Track layouts and section lengths for case studies 4, 5 and 6 are shown in Figure 8. It should be noted that circles are input-output location.

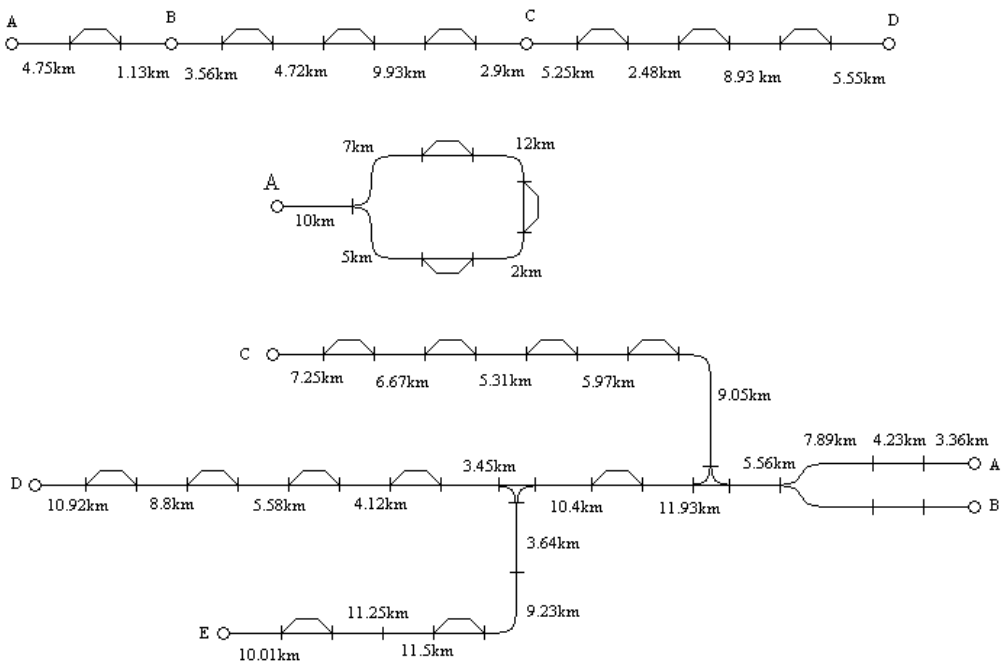



**Figure 8.** Track layouts for case study 4, 5 and 6

The other examples are serial lines consisting of alternating section and passing loop segments. The section lengths for case studies 1, 2 and 3 are shown below. The lengths of passing loops are underlined. The last case (i.e. 7) is case 1, 2 and 3 added together respectively. At the meeting of the sub problems there are two adjacent sections not separated by a passing loop.

Case 1: 6.753, 0.847, 4.732, 0.866, 6.956, 0.83, 4.137, 0.88, 4.745, 0.872, 4.083, 0.548, 4.767, 1.138, 2.879, 0.865, 2.558, 0.867, 3.167, 0.932, 7.608, 0.96, 3.924, 0.846, 5.38, 1.08, 6.036, 1.068, 7.732, 0.827, 4.852, 1.094, 2.7219, 1.8411, 9.639, 0.864, 6.23, 0.923, 4.618

Case 2: 2.725, 0.982, 8.266, 1.19, 4.956, 0.874, 13.985, 0.838, 6.129, 0.834, 5.914, 1.042, 5.118, 0.868, 7.654, 0.666, 6.094, 1.029, 6.879, 0.876, 4.747, 0.742, 5.591

Case 3: 14.252, 0.874, 9.308, 0.84, 3.796, 0.906, 8.467, 0.859, 7.074, 0.929, 8.533, 0.868, 9.589, 0.886, 7.789, 0.864, 10.8, 1.445, 0.997, 0.606, 6.824, 0.886, 14.618, 0.844, 8.058, 0.84, 15.589, 0.981, 11.249, 0.754, 10.239, 1.125, 9.563, 0.935, 11.062, 0.867, 12.339, 0.917, 10.622, 1.03, 13.357, 1.095, 19.609, 0.846, 14.161

### 5.2 Lower Bound

The following lower bound for the makespan is proposed and is used to judge the quality of solutions.

$$LB = \max_{j \in M} \; lb_j \;\; , \; lb_j = \sum_{o_{i,k} \in U_j} \; \bar{p}_{i,k} \; + \min_{o_{i,k} \in U_j} \; \tau_{i,k} \; + \min_{o_{i,k} \in U_j} \; \phi_{i,k} \qquad \forall j \in M \tag{18}$$

$$\bar{p}_{i,k} = p_{i,k} + \delta_{i,k} + lag_{i,k} \; , \; \forall o_{i,k} \in O \tag{19}$$

$$\tau_{i,1} = rlt_i \; , \; \tau_{i,k} = \tau_{i,k-1} + p'_{i,k-1} \left[ = \sum_{k'=1,..,k-1} \; p'_{i,k'} \; \right] \quad k = 2,..,K_i \tag{20}$$

$$\phi_{i,K_i} = 0 \; , \;\; \phi_{i,k} = \phi_{i,k+1} + p''_{i,k+1} \left[ = \sum_{k'=k+1..,K_i} \; p''_{i,k'} \; \right] \quad k = K_i - 1,..,1 \tag{21}$$

$$p'_{i,k} = p_{i,k} + \delta_{i,k} \; , \;\; p''_{i,k} = p_{i,k} + lag_{i,k} \tag{22}$$

For job *i*, $\phi_{i,k}$ is the minimum time left after $o_{i,k}$ is performed, and $\tau_{i,k}$ is the minimum time required before $o_{i,k}$ can be performed. $\tau_{i,k}$ is associated with the path of the front, while $\phi_{i,k}$ is associated with the path of the rear. The lower bound is computed as the maximum machine lower bound. A lower bound associated with a machine is the total processing requirement for that machine plus the minimum time to reach the machine and leave the machine for a particular job.

The main limitation of the proposed lower bound is that setups (headways) between operations of different jobs have not been included. For problems without setups the lower bound will be more accurate.

### 5.3. Results

The full list of numerical results is too large to be displayed in this paper so a summary of the best results is displayed. The results of the constructive algorithm for forward / backward scheduling and for the buffer representation option are shown in Table 2 and 3. The solutions obtained with the BOV permit option contain a reasonable number of BOV's and often have significantly smaller makespans. In Table 2 and 3 the results for example 1-6 were obtained in under 10 seconds. Example 7 however is a much larger problem and requires in the order of 10 minutes. For the forbid BOV option and forward scheduling, example 7 was too large to solve in entirety in reasonable time and only 20 of the 30 trains were scheduled.

**Table 2.** Constructive algorithm starting solutions (forward scheduling case)

| Case | LB | Insert Greedy / BOV forbid | Insert Greedy / BOV permit | | Insert Random / BOV forbid | Insert Random / BOV permit | |
|---|---|---|---|---|---|---|---|
| | | Cmax | Cmax | BOV | Cmax | Cmax | BOV |
| 1 | 144.64 | 367.32 | 235.8 | 9 | 1021.57 | 1169.05 | 69 |
| 2 | 201.1 | 507.98 | 232.57 | 17 | 791.69 | 911.95 | 59 |
| 3 | 298.1 | 504.06 | 457.58 | 6 | 2148.98 | 2461.39 | 78 |
| 4 | 118.35 | 179.86 | 149.49 | 8 | 358.18 | 362.41 | 20 |
| 5 | 162.8 | 212.25 | 165.05 | 28 | 272.6 | 349.2 | 0 |
| 6 | 309.1 | 1026.96 | 554.2 | 74 | 1999.06 | 1941.27 | 35 |
| 7 | 442.92 | - | 749.27 | 10 | 5221.62 | 6250 | 296 |

**Table 3.** Constructive algorithm starting solutions (backward scheduling case)

| Case | LB | Insert Greedy / BOV forbid | Insert Greedy / BOV permit | | Insert Random / BOV forbid | Insert Random / BOV permit | |
|---|---|---|---|---|---|---|---|
| | | Cmax | Cmax | BOV | Cmax | Cmax | BOV |
| 1 | 144.64 | 249.35 | 235.8 | 18 | 1013.48 | 1177.04 | 69 |
| 2 | 201.1 | 274.9 | 232.57 | 15 | 785.63 | 929.38 | 53 |
| 3 | 298.1 | 473.32 | 457.58 | 9 | 2153.01 | 2462.45 | 76 |
| 4 | 118.35 | 143.95 | 144.88 | 6 | 341.22 | 331.9 | 26 |
| 5 | 162.8 | 198.2 | 164.8 | 27 | 332.55 | 315.1 | 1 |
| 6 | 309.1 | 528.53 | 554.2 | 69 | 1873.85 | 1901.31 | 31 |
| 7 | 442.92 | 901.15 | 749.27 | 33 | 5123.32 | 6270.48 | 355 |

A summary of the extensive meta-heuristic results are shown in Table 4.

**Table 4.** Summary of meta-heuristic solutions

| Case | LB | Best (Forward) | | Best (Backward) | |
|---|---|---|---|---|---|
| | | SA | LS | SA | LS |
| 1 | 144.64 | 205.14 | 208.468 | 226.42 | 236.47 |
| 2 | 201.1 | 239.24 | 229.675 | 229.07 | 227.7 |
| 3 | 298.1 | 410.774 | 411.159 | 441.91 | 444.42 |
| 4 | 118.35 | 137.2 | 146.61 | 143.95 | 143.95 |
| 5 | 162.8 | 196.15 | 198.1 | 192.8 | 196.4 |
| 6 | 309.1 | 975.5 | 888.352 | 508.99 | 508.99 |
| 7 | 442.92 | 859.58 | 792.64 | 900.00 | 755.12 (11 BOV) |

LS was found to be slightly superior particularly on larger problems like example 7. LS is able to resolve the original BOV in the starting solution more easily. The best solutions were also obtained in the majority of cases when forward scheduling. This is quite surprising since backward scheduling gave superior solutions via the constructive algorithm.

The fourth parameter set was generally the best for SA. This is not surprising since more evaluations were performed. Starting from a low temperature (a fairly greedy approach) has previously been observed to give better solutions and has again been observed here. LS is generally best when the number of iterations is highest, and hence 3000 iterations was best.

For SA the best perturbation operator was generally SA3 (the critical path operator) however large differences in solution quality were not often observed. For LS the best permutation operators were clearly LS3 and LS4. This is not surprising since more effort is expended by these operators. LS3 was observed to be fractionally better than LS4.

The overall strategy of constructing a feasible solution with the CA and insert greedy & forbid BOV option and then refining the solution by MH's using the forbid BOV option was observed to be generally the best. This is not entirely surprising since starting from the best constructive algorithm solution and penalising buffer occupancy violations will always give a good quality solution that is always feasible. Starting from a randomly generated solution of poor quality and which is feasible or infeasible is not particularly efficient. In fact, those strategies regularly gave very poor solutions. This is because the effort required by the meta-heuristic algorithms was just too great when starting the search so far away from the optimal and for a problem with such a complex search space.

Alternative strategies that permit BOV's but penalise their occurrence however could often be clearly superior to the strategy mentioned in the previous paragraph. The ability to move through the infeasible part of the search space has definite merit. For example, movement through the feasible part of the search space can limit the solution quality. This is because there is sometimes no feasible move that can be made without causing BOV's. Alternatively all of the feasible moves that can be made may result in greatly inferior solutions. Starting from essentially the classical job shop solution (with infinite buffers) and proceeding to resolve the existing buffer occupancy violations (BOV's) is quite efficient and useful. The main downside however is that solution feasibility can never be truly guaranteed. In the majority of cases these strategies performed very well but in some cases the results were highly infeasible. The number of buffer occupancy violations that still needed to be resolved was quite large. Re-running the meta-heuristic again from the current infeasible solution is the best avenue for resolving all of the BOV's.

The CPU time required by the meta-heuristics was very reasonable. The largest time required on the largest problem was about 10 minutes. Most of the computation time is spent evaluating the schedule makespan via the disjunctive graph. This is a well known disadvantage of job shop sequencing techniques however further modification and simplification can still be made and will speed up the process.

In general the solution quality was fairly good. For the smaller problems the optimal schedules could be obtained. On the larger serial instances solution quality was relatively close to the lower bounds however further improvements could still be obtained. Some manual manipulation of these solutions allowed some even better schedules to be obtained. On the more complex network example (i.e. example 6) solution quality was however quite poor. The reason for this is explained in the next paragraph.

Solution quality was significantly poorer on problems that contain adjacent sections not separated by crossing loops (i.e. example 6). This is because the necessary compound moves have not yet been implemented. Many of the current unitary moves must be restricted because precedence impossibilities result. Consequently solutions that are closer to the optimal can not always be reached. Solution quality may also be poor when unnecessary overtaking is

allowed. Undoing unnecessary overtaking is difficult to achieve by standard operators and may be more computationally demanding. We are certain that the restriction of unnecessary overtaking in the meta-heuristics and in the constructive algorithm will lead to even better solutions.

Simulated Annealing was re-applied using the compound interchange as the primary perturbation operator. Table 5 compares the new results with the previous ones and shows that significant improvements have been realised. The speed of convergence is also much more rapid though this is not shown here. It should also be noted that these solutions are particularly good considering the fact that buffer occupancy violations (BOV) were not permitted. Allowing BOV's has been a useful mechanism to "get out of trouble" when parts of the schedule become highly congested and simple interchanges become insufficient at feasibly perturbing the solution. With compound moves the reliance to use this option is much reduced though at times it has been observed that it is still beneficial to temporarily allow moves that cause BOV's.

**Table 5.** Comparison of solutions obtained by SA with compound perturbation operator

| Case | LB | Previous Best | New Best | Improvement |
|---|---|---|---|---|
| 1 | 144.64 | 202.92 | 183.40 | 19.52 |
| 2 | 201.1 | 220.22 | 214.75 | 5.47 |
| 3 | 298.1 | 404.41 | 398.98 | 5.43 |
| 4 | 118.35 | 136.54 | 127.9 | 8.64 |
| 5 | 162.8 | 196.15 | 185.99 | 10.16 |
| 6 | 309.1 | 494.95 | 469.62 | 25.33 |
| 7 | 442.92 | 792.64 | 701.66 | 90.98 |

## 6. Conclusions

This paper addressed the representation and construction of accurate train timetables by a hybrid job shop approach. **As a consequence of this research a new approach is proposed for solving job shop problems with capacitated buffers.** The job shop approach in particular was developed because it has many advantages over existing conventional approaches and has the potential to be greatly extended. Unique aspects of train scheduling such as train length, dwell times, blocking, headways, alternative routing, re-entrant and circular path were incorporated into the job shop scheduling framework and more precisely into an innovative AON disjunctive graph representation. To our knowledge these unique components have not been incorporated in this way before. The proposed AON graph representation differs from traditional versions in several important ways. For example the disjunctive arcs are no longer the reverse of one another and do not have a weight of zero. The weights which represent time lags or overlaps between operations now include train headways, dwell times and processing times and can take either positive or negative values. The immediate successor operation is also no longer sufficient to signify the completion and departure of an operation and the availability of a machine, and consequently arc generation is more sophisticated.

With an efficient graph representation of the solution, algorithms for the construction of a feasible schedule were then developed. The main algorithm constructs a schedule job by job and operation by operation using insertion, backtracking and sophisticated dynamic route selection mechanisms. In this way a feasible solution is guaranteed. The purpose of the constructive algorithm (CA) was to provide a robust and substantial search capability as

apposed to the alternative of obtaining any solution as quick as possible using dispatching rules.

Simulated Annealing and Local Search meta-heuristic improvement approaches were lastly adapted / extended. Train schedules in particular are perturbed using shift moves or an innovative compound interchange operation. Solution feasibility may be strictly enforced or infeasible moves may be permitted and penalised in the objective function. Either way good quality solutions approaching optimality can be quickly obtained from starting solutions obtained via constructive algorithms.

From the numerical investigations and case study it was observed that ignoring BOV's allows significantly higher quality solutions to be constructed more quickly although the solutions are mostly infeasible. ***The application of meta-heuristics may be (but not exclusively) viewed as the primary mechanism in which to resolve the original BOV's***.

Movement through only the feasible part of the search space can limit the solution quality. Sometimes there are very few if any conventional moves that can be made without causing BOV's. Similarly all of the feasible conventional moves that can be made may result in greatly inferior solutions. Movement through the infeasible part of the search space however does not guarantee convergence to better (overall) feasible solutions.

Solution quality is inferior on problems that contain adjacent sections not separated by crossing loops. This is because the necessary compound moves have not yet been implemented. Many of the current unitary moves must be restricted because precedence impossibilities result and consequently solutions that are closer to the optimal can not always be reached.

Compound perturbation operators can restrict multiple overtaking conflicts from occurring. The properties upon which critical path operators have been applied previously do not hold for train scheduling problems. Critical path operators may therefore be inferior or insufficient because the movement of non critical operations associated with capacitated buffers can affect critical operations. Even so the numerical investigation has shown that critical path operators still perform adequately if not more so.

During the course of this research a number of alternative avenues and extensions were found for further research. These include the merging and splitting of trains, the minimisation of disruption when additional trains must be added to an existing schedule, scheduling under alternative objective criteria, scheduling involving machines with periods of unavailability, scheduling involving fixed jobs.

**Acknowledgements**

This research was funded by Queensland Rail and the ARC.

## REFERENCES

**Adenso-Diaz B., Gonzalez M. O., Gonzalez-Torre P. (1999),** On line timetable re-scheduling in train services. *Transportation Research Part B*, 33, 387-398.

**Allahverdi A., Gupta J. N. D. and Aldowaisan T. (1999**), A review of scheduling research involving setup considerations. *Omega, International Journal of Management Science*, 27, 219-239.

**Billionnet A. (2003),** Using integer programming to solve the train platforming problem. *Transportation Science*, 37(2), 213-222.

**Brannlund U., Lindberg P. O. Nou A. Nillson J.-E. (1998),** Railway timetabling using lagrangian relaxation. *Transportation Science*, 32, 358-369

**Burdett R. and Kozan E. (2004),** Absolute capacity determination and timetabling in railways. Proceedings of the Fifth Asia Pacific Industrial Engineering and Management Systems Conference**,** APIEMS 2004.
**Burdett R. and Kozan E. (2006),** Techniques for Absolute Capacity Determination in Railways. *Transportation Research B*, 40(8), 616-632.
**Cai X., and Goh C. J. (1994).** A fast heuristic for the train scheduling problem. *Computers and Operations Research*, 21(5), 499-510.
**Cai X., Goh C. J., and Mees A. I. (1998).** Greedy heuristics for rapid scheduling of trains on a single track. *IIE Transactions*, 30, 481-493.
**Candido M. A. B., Khator S. K. and Barcia R. M. (1998),** A genetic algorithm based procedure for more realistic job shop scheduling. *International Journal of Production Research*, 36(12), 3437-3457.
**Carey M. and Lockwood D. (1992),** A model, algorithms and strategy for train pathing. Research Report, Faculty of Business and Management, University of Ulster, North Ireland. *J Operational Res. Soc.*
**Carey M. (1994),** A model and strategy for train pathing with choice of lines, platforms, and routes. *Transportation Research B*, 28(5), 333-353.
**Carey M. (1994),** Extending a train pathing model from one-way to two-way track. *Transportation Research B*, 28(5), 395-400
**Carey M and Carville S. (2003),** Scheduling and platforming trains at busy complex stations. *Transportation Research A*, 37, 195-224.
**Chiang T. W., Hau H. Y., Chiang H. M., Ko S. Y., and Hsieh C. H. (1998),** Knowledge based system for railway scheduling. *Data and Knowledge Engineering*, 27, 289-312.
**Cordeau J. F., Toth P., Vigo D. (1998).** A survey of optimisation models for train routing and scheduling. *Transportation Science*, 32, 380-404
**Corry P. and Kozan E. (2004),** Job scheduling with technical constraints. *Journal of the Operational Research Society*, 55, 160-169.
**Daniels R. L, Hua S. Y., and Webster S. (1999),** Heuristics for parallel machine flexible resource scheduling problems with unspecified job assignment. *Computers and Operations Research*, 26, 143-155
**Dorfman M. J. and Medanic J. (2004),** Scheduling trains on a railway network using a discrete event model of railway traffic. *Transportation Research B*, 38, 81-98.
**Goverde R. M. P. (1999),** Improving punctuality and transfer reliability by railway timetable optimisation. P. Bovy, ed. *Proc. TRAIL 5th Annual Congress*, Vol. 2. Delft, The Netherlands.
**Groflin H. and Klinkert A. (2007),** Feasible insertions in job shop scheduling, short cycles and stable sets. . *European Journal of Operational Research*, 177, 763-785
**Higgins A., Kozan E., and Ferreira L. (1996),** Optimal scheduling of trains on a single line track. *Transportation Research B*, 30(2), 147-161.
**Higgins A., Kozan E., and Ferreira L. (1997).** Heuristic techniques for single line train scheduling. *Journal of Heuristics*, 3, 43-62.
**Khosla I. (1995),** The scheduling problem where multiple machines compete for a common buffer. *European Journal of Operational Research*, 84, 330-342
**Kim D. W., Na D. G., and Chen F. (2003),** Unrelated parallel machine scheduling with setup times and a total weighted tardiness objective. *Robotics and Computer Integrated Manufacturing*, 19, 173-181
**Kim Y. K., Park K., and Ko J. (2003),** A symbiotic evolutionary algorithm for the integration of process planning and job shop scheduling. *Computers and Operations Research*, 30, 1151-1171
**Kis T. (2003)**, Job shop scheduling with processing alternatives. *European Journal of Operational Research*, 151, 307-332.

**Kozan E. and Burdett R. (2005),** A Railway Capacity Determination Model and Rail Access Charging Methodologies. *Transportation Planning and Technology*, 28(1), 27-45
**Kroon L. G., Romeijn H. E., Zwaneveld P. J. (1997),** Routing trains through railway stations: complexity issues. *European Journal of Operational Research*, 98, 485-498.
**Kroon L. G., and Peeters W. P. (2003),** A variable trip time model for cyclic railway timetabling. *Transportation Science*, 37(2), 198-212 (2003).
**Kolonko M. (1999),** Some new results on simulated annealing applied to the job shop scheduling problem. *European Journal of Operational Research*, 113, 123-136
**Lindner T. (2000),** Train schedule optimisation in public rail transport. Ph.D. thesis, Technical University Braunschweig, Braunschweig, Germany.
**Mascis A., and Pacciarelli D. (2002),** Job-shop scheduling with blocking and no-wait constraints. *European Journal of Operational Research*, 143, 498-517
**Mastrolilli M. and Gambardella L. M. (2000)**, Effective neighbourhood functions for the flexible job shop problem. *Journal of Scheduling*, 3, 3-20
**Mattfeld D. C. and Bierwirth C. (2004),** An efficient genetic algorithm for job shop scheduling with tardiness objectives. *European Journal of Operational Research*, 155, 616-630.
**Murovec B. and Suhel P. (2004),** A repairing technique for the local search of the job shop problem. European Journal of Operational Research, 153, 220-238.
**Nawaz M., Enscore E. E., and Ham I. (1983),** A heuristic algorithm for the m-machine n-job flow shop sequencing problem. *Omega*, 11(1), 91-95
**Nowicki E. (1999),** The permutation flow shop with buffers: a tabu search approach. *European Journal of Operational Research*, 116, 205-219
**Odijk M. A. (1996),** A constraint generation algorithm for the construction of periodic railway timetables. *Transportation Research*, 30, 455-464.
**Sahin I. (1999).** Railway traffic control and train scheduling based on inter-train conflict management. *Transportation Research B*, 33, 511-534.
**Steinhofel K., Albrecht A., and Wong C. K. (1999),** Two simulated annealing based heuristics for the job shop scheduling problem. *European Journal of Operational Research*, 118, 524-548.
**Van Laarhoven P. J. M., Aarts E. H. L., Lenstra J. K. (1992),** Job shop scheduling by simulated annealing. *Operations Research*, 40(1), 113-125
**Werner F. and Winkler A. (1995),** Insertion techniques for the heuristic solution of the job shop problem. *Discrete Applied Mathematics*, 58, 191-211.
**Zwaneveld P. J., Kroon L. G., Zoghby J., Barnes J. W. and Hasenbein J. J. (2005),** Modelling the re-entrant job shop scheduling problem with setups for meta-heuristic searches. *European Journal of Operational Research*, 167, 336-348.
**Zwaneveld P. J., Kroon L. G., Romeijn H. E., Salomon M., Dauzere Peres S., Stan C. P. M. van Hoesel and Ambergen H. W. (1996).** Routing trains through railway stations: model formulation and algorithms. *Transportation Science*, 30, 181-194
**Zwaneveld P. J., Kroon L. G., Van Hoesel S. P. M. (2001),** Routing trains through a railway station based on a node packing model. *European Journal of Operational Research*, 128, 14-33.

## Appendix

**Algorithm 1:** ***CreateCompoundInterchange***( $o_{i,k}, o_{i',k'}, cpdint$)

**Begin**

$common_{i.i'} = \{ j \in M \mid \exists k, k' : m_{i,k} = m_{i,k'} = j \}$ ; // Define the set of common machines

// Determine the order in which common machines are visited by either job:

***GetOrdering*** $i, i', common_{i,i'}, same\_dir, order_i, order_{i'}$ ;

Determine position of $o_{i,k}$ & $o_{i',k'}$ in the orderings:

$pos_{i,k} = k^* \quad s.t. \quad 1 < k^* < |order_i| \ \wedge \ order_{i,k^*} = o_{i,k}$ ;

$pos_{i',k'} = k^* \quad s.t. \quad 1 < k^* < |order_{i'}| \ \wedge \ order_{i',k^*} = o_{i',k'}$ ;

// Identify the compound move:

*found* = ***IdentifyPassing***(*same_dir, passPos*, $order_i, order_{i'}$ ); // Identify whether trains pass each other

**if**(*found*)

**begin**

**if**(*same_dir*) ***UndoPassing***(*same_dir, passPos,* $order_i, order_{i'}$ , *cpdint*);

**else** ***ShiftPassing***(*same_dir,* $pos_{i,k}$ ,*passPos*, $order_i, order_{i'}$ , *cpdint*);

**end**

**else** ***MakePassing***(*same_dir,* $pos_{i,k}$ , $order_i, order_{i'}$ , *cpdint*);

**End**

**Algorithm 2:** ***IdentifyPassing***(*pos,* $order_i, order_{i'}$ )

**Begin**

$z = 0; \quad prev = 0; \quad ub = |order_i|$ ;

*found* = false; // Assume no overtake until otherwise proven

**while** $z < ub$ and $\neg found$ **begin**

$o_{i,k} = order_{i,z}$ ; $o_{i',k'} = \ same\_dir \ ? order_{i',z} : order_{i',ub-1-z}$ ;

$j = m_{i,k} = m_{i',k'}$ ;

**if** $\neg isbuffer \ j$ **begin**

$curr = \ o_{i,k} \prec o_{i',k'} \ ? 1 : -1$;

***if*** $z > 0$ and $prev \neq curr$ **begin** // An overtake has been found

*found* = true; // Update the flag

*pos* = z-1; // Define the overtake position

**end**

*prev* = *curr*;

**end**

*z*++;

**end**

**return** *found*;

**End**

**Algorithm 3:** ***MakePassing***(*same_dir, pos,* $order_i, order_{i'}$ ,*cpdint*)

**Begin**

$z = pos; \quad ub = |order_i|$ ;

**while** $z < ub$ **do begin**

$o_{i,k} = order_{i,z}$ ; $o_{i',k'} = \ same\_dir \ ? order_{i',z} : order_{i',ub-1-z}$ ; $j = m_{i,k} = m_{i',k'}$ ;

**if** $\neg isbuffer\ \ j \quad cpdint.\Theta \cup \ o_{i,k}, o_{i',k'}$ ; // Add element (i.e. 2 tuple) to the set

$z$++; // Go to the next pair

**end**

**End**

**Algorithm 4:** ***UndoPassing***($pos$, $passPos$, $order_i$, $order_{i'}$, $cpdint$)

**Begin**

$z = passPos; \quad ub = |order_i|;$

$incr = \ pos < passPos \ ? -1 : 1;$

**do begin**

$o_{i,k} = order_{i,z}; \ o_{i',k'} = \ same\_dir \ ? order_{i',z} : order_{i',ub-1-z}; \ j = m_{i,k} \ = m_{i',k'} \ ;$

**if** $\neg isbuffer\ \ j \quad cpdint.\Theta \cup \ o_{i,k}, o_{i',k'}$ ; // Add element (i.e. 2 tuple) to the set

$z$+=$incr$; // Go to the next pair

**end**

**while** $\ incr = 1 \wedge z < ub \ \vee \ incr = -1 \wedge z \geq 0$

**End**

**Algorithm 5:** ***ShiftPassing***($pos$, $passPos$, $order_i$, $order_{i'}$, $cpdint$)

**Begin**

$z = pos; \quad ub = |order_i|;$

$incr = \ pos < passPos \ ? 1 : -1;$

**do begin**

$o_{i,k} = order_{i,z}; \ o_{i',k'} = \ same\_dir \ ? order_{i',z} : order_{i',ub-1-z}; \ j = m_{i,k} \ = m_{i',k'} \ ;$

**if** $\neg isbuffer\ \ j \quad cpdint.\Theta \cup \ o_{i,k}, o_{i',k'}$ ; // Add element (i.e. 2 tuple) to the set

$k$+=$incr$; // Go to the next pair

**end**

**while** $\ incr = 1 \wedge k < passPos \ \vee \ incr = -1 \wedge k > passPos$

**End**