Subgraph ejection chains and tabu search for the crew scheduling problem

L Cavique[1], C Rego[2] and I Themido[3]

[1]*Instituto Politécnico, EST, Setúbal, Portugal,* [2]*Faculdade de Ciências, DEIO, Lisboa, Portugal and* [3]*Instituto Superior Técnico, CESUR, Lisboa, Portugal*

The tabu search algorithms for the Crew Scheduling Problem (CSP) reported in this paper are part of a decision support system for crew scheduling management of the Lisbon Underground. The CPS is formulated as the minimum number of duties necessary to cover a pre-defined timetable under a set of contractual rules. An initial solution is constructed following a traditional run-cutting approach. Two alternative improvement algorithms are subsequently used to reduce the number of duties in the initial solution. Both algorithms are embedded in a tabu search framework: *Tabu-crew* takes advantage of a form of *strategic oscillation* for the neighbourhood search while the *run-ejection* algorithm considers compound moves based on *a subgraph ejection chain method.* Computational results are reported for a set of real problems.

Keywords: crew scheduling; ejection chains; heuristics; run-cutting; tabu search

Introduction

The planning and control of the operations of the Lisbon Underground (LU) forms a complex decision process which requires the solution of a number of related problems. The algorithms reported in this paper are part of a decision support system for crew scheduling management. The implementation of this system was considered a priority as the Underground network is undergoing extensive alterations and expansion including the introduction of several new lines in the next few years. In this new context the traditional manual crew scheduling methods, improved over many years for the same service, are presently unable to cope.

Over the last two decades, considerable attention has been devoted to the scheduling of vehicles and their crews, and the number of practical applications has risen. The most important results obtained since 1975 are compiled in the seven volumes of papers on Computer-Aided Scheduling of Public Transport edited by Bodin and Bergman,[1] Wren,[2] Rousseau,[3] Daduna and Wren,[4] Desrochers and Rousseau,[5] Daduna *et al*[6] and Wilson.[7]

There are three main approaches reported in the literature for the Crew Scheduling Problem (Bodin *et al*[8] and Wren and Rousseau[9]): the run-cutting heuristic, the matching algorithm and the set covering formulation. The run-cutting heuristic was used in the 1970s, in the American RUCUS package and in the algorithm developed by Parker

and Smith for the British TRACS system; it is a constructive algorithm that emulates the procedures used by manual schedulers. The matching algorithm is used in RUCUS-II and in an initial version of HASTUS; the method is divided into three parts: partition of the timetable into blocks, graph generation and duty achievement. The set covering formulation is used in the package IMPACS of the BUSMAN system and in the Crew-Opt package; the latter included in an early version of HASTUS. The method first generates a large set of feasible duties and, in a second stage, finds the minimal covering set.

The most widely used decision support systems for Crew Scheduling in the Public Transport Sector are probably those of HOT from Germany, BUSMAN from the United Kingdom and HASTUS from Canada. Some unpublished systems such as Teleride-Sage and UMA, mentioned by Wren and Rousseau,[9] are also common.

The scheduling of the Lisbon Underground's crews is a traditional Crew Scheduling Problem (CSP) which can be formulated as the minimum number of duties, that satisfies a number of contractual rules, necessary to cover a pre-defined train timetable. Before presenting the problem constraints, some terms used in the CSP literature will be introduced, in the context of the LU case study, to help the reader. For a full glossary of terms in vehicle and crew scheduling refer to Hartley.[10]

The terms 'trip' and 'block' are closely related to the train timetable, while the terms 'piece of work' (or 'piece') and 'duty' (or 'run') are related to the crew schedule:

- *timetable*—is a set of n trips, $T = \{t_1, t_2, t_3, \ldots, t_n\}$, that almost corresponds to the public timetable. The trains run

Correspondence: Dr I Themido, Av. Rovisco Pais, 1096 Lisboa Codex, Portugal.
E-mail: ithemido@alpha.ist.utl.pt

according to the timetable and each one is identified by a number.

- *trip*—is a one way movement of a train between two terminal points, the smallest period (or elementary crew activity) into which the timetable can be divided. A trip has five attributes: train number, starting place and time, finishing place and time. The relief points must coincide with either the start or the end of a trip, and never happen during the trip itself. In the LU case relief points are at the track terminus, and trips last between 20 and 40 min.

- *block*—is the set of all trips produced by the same train i, $b = \{t_p, t_{p+1}, \ldots, t_q\}$, and $B = \{b_1, b_2, \ldots, b_m\}$ the set of all blocks or trains. When trips produced by the same train are interrupted, for instance during off-peak-hours, and divided into two or more groups, we say that there is a broken block.

- *piece of work (or piece)*—is a set of consecutive trips in a block, covered by the same crew, under certain contractual rules. A given piece j is defined as $p_j = \{t_r, t_{r+1}, \ldots, t_s\}$, and $P = \{p_1, p_2, \ldots, p_k\}$ is the set of all pieces. The block partition is a set of non-overlapping pieces of work that exactly covers the block.

- *duty (or run)*—is the work to be performed by a crew in one day. In the LU case most duties have two pieces of work separated by a meal break $d_1 = \{p_i, p_j\}$. But duties with one single piece of work, $d_1 = \{p_i\}$, thereby leaving the remaining time free to act as reserve crews, are also possible. Let $D = \{d_1, d_2, \ldots, d_3\}$ denote the set of all duties which can be constructed with a pre-defined set of pieces P, and let S be the set of duties in the problem solution. In the LU case as all duties have an identical cost, duties with one single piece of work are considered, on average, less efficient than those with two pieces of work that cover more trips for the same cost.

Contractual and operational rules can be organized under the following headings:

- *relief points*—must be the same at the beginning and end of the duties and meal breaks.
- *pieces of work*—maximum of 3 1/2 h and minimum of 1 1/2 h.
- *duty*—total duration of 9 h, separated into two working periods by a 1 h meal break. A complete crew duty includes one or two pieces of work, the meal break, the report and clear time and reserve periods to make up the 9 h. There is a maximum of 5 and a minimum of 3 h for each working period. The report time and clear time must be at least 20 min. Periods of work must start and finish on the hour or, at 15, 30, or 45 min past the hour.
- *meal break*—the meal break is 1 h and it must coincide with the period of functioning of the canteen (11:00–15:00 and 17:00–22:00), except for the duties which terminate between 14:00–14:30 and 21:00–21:30.

A set of real train timetables was used in order to test the performance of the algorithms developed. The problem sizes range between 528 and 718 trips, total crew workloads vary between 280 and 386 hours and the number of blocks between 21 and 26 (equal to the number of trains in circulation). In the near future, it is expected that timetables will have more than a thousand trips, and that additional constraints will have to be considered.

The purpose of this paper is to describe new heuristic approaches for the CSP as the dimension of the problem recommends an heuristic approach.[11] The initial solution is constructed from scratch using a run-cutting approach. Two alternative improvement algorithms are also presented and in both cases a tabu search procedure is used to reduce the number of duties in the initial schedule. *Tabu-crew* takes advantage of a form of *strategic oscillation* for the neighbourhood search while the *Run-ejection* algorithm considers compound moves based on *a subgraph ejection chain method*.

The remainder of this paper is organised as follows. Firstly, the constructive algorithm for generating initial schedules is described. Then, the improvement algorithms, *Tabu-crew* and *Run-ejection* are presented followed by computational results. Finally, a summary and concluding remarks are presented.

Generating initial schedules: the run-cutting algorithm

To obtain an initial feasible crew schedule a run-cutting algorithm was developed. The run-cutting method is a traditional approach inspired by manual schedule procedures that can be implemented in different ways to take advantage of the structure of the particular problem.[8]

With this constructive method the timetable is progressively covered with duties (with one or two pieces of work), until the cover is complete. During this process a trip is covered if a duty (or crew) is assigned to it, and uncovered otherwise. The algorithm generates a feasible run (or duty) by cutting (or selecting) uncovered trips from a block of the timetable and covering the trips with that new duty. This process is repeated until all trips are covered with duties. The algorithm has two embedded loops: the internal loop creates a list of feasible duties, called the *candidate list*, C, while the external loop selects the best duty of the list and covers the timetable with it.

Procedure: *Run-cutting*
  Assign a work timetable $W = T$
  Assign the set of all duties $S = \varnothing$
  Generate a feasible duty d_1 from W
  While $W \neq \varnothing$ and it is possible to generate a feasible duty d_1, do
      Set $S = S \cup \{d_1\}$
      Set $W = W \backslash \{d_1\}$
      Generate a candidate list C of feasible duties d_1^C from W
      Select the best duty d_1 from C
  endwhile
  Return S

We recall that duties are formed by pieces of work. In order to obtain a good crew schedule it is important to start with appropriate pieces of work. The quality of a piece depends on the number of trips covered (the more the better) and on the room left for subsequent adjacent pieces. An inadequate piece would leave a few adjacent trips, insufficient to complete a new piece (in this case at least one and a half hours as stated in the rules mentioned in the introduction), putting in jeopardy the total cover of the train timetable.

To make sure that all pieces must either not leave adjacent trips uncovered or leave enough room for future pieces, a penalty function was introduced where nt denotes the number of trips adjacent to each side of a piece, and lb, is a lower bound for the number of uncovered trips adjacent to a piece:

Penalty function

(i)   0   if $nt = 0$
(ii) $1/nt$ if $lb \leqslant nt \leqslant \infty$
(iii) ∞ if $nt < lb$

This penalty function applies to both sides of a piece, being deducted from the number of trips the piece can cover to produce the *piece evaluation function*. The duty evaluation function, used to select the best duty, d_1, from the candidate list is equal to the sum of the evaluation functions for the pieces it contains.

A drawback of the run-cutting algorithms is their lack of flexibility.[9] Some authors note that it is often difficult to adapt the algorithm to a different set of constraints. In order to overcome these difficulties, we consider a *generate-and-test strategy*[12] based on two independent procedures: a generator of combinatorial objects and a checker of contractual rules. With this architecture the algorithm can easily be adapted to incorporate additional constraints.

Another limitation of the run-cutting algorithms is their difficulty to cope with problems where break times must be specified in the runs.[9] To overcome this difficulty we consider combinatorial objects generated in a systematic form. By establishing an analogy with a lexicographic search,[13] the generation of a duty may be viewed as the creation of a sub-word, of a larger word representing a sequence of all the trips in the timetable.

Guiding the run-cutting procedure using tabu search: the tabu-crew algorithm

Tabu-crew, used in the improvement phase, is an iterative procedure which attempts to reduce the total number of duties using tabu search. Tabu search is a meta-heuristic that guides a local heuristic search procedure throughout the solution space beyond local optimality. The method, introduced by Glover,[14] is based on ideas derived from classical optimisation techniques such as surrogate constraint methods, cutting plane approaches, steepest ascent/mildest methods and some concepts of artificial intelligence. In spite of this, tabu search can be considered a recent meta-heuristic technique when compared with alternative approaches such as simulated annealing and genetic algorithms (see Glover and Laguna[15] for a comprehensive description of the method and applications).

For readers who are not familiar with tabu search, we will describe the basic principles of the method and some components used in our algorithms.

The method is an iterative search procedure whereby the algorithm *moves* from one solution to another, in a defined neighbourhood structure, avoiding being trapped in local optima. Broadly speaking, tabu search lies on the use of flexible memory structures which impose restrictions on the search process guiding it to 'good' regions of the solution space. Generally, such restrictions are related to the problem setting and represent some kind of interdictions (tabu restrictions) on the solution attributes (which may be forbidden to enter new solutions in subsequent iterations of the method). Tabu restrictions can be used with different levels of strictness depending on the type of memory under consideration: *short-term memory* is generally used to prevent the method from returning to solutions already visited while *longer-term memory* plays a dual role either focusing the search into regions of high quality solutions, intensifying the search, or guiding the method to other directions diversifying the search to new regions of the solution space.

From the practical standpoint, short term memory is usually implemented using a *tabu list* made up of a set of (reverse) moves which are forbidden for a certain number of iterations, the *tabu tenure*. Longer-term memory is usually based on the frequency of some move attributes and restrictions are imposed as penalty factors to the objective function in order to modify the choice rules for the moves. In addition, strategic oscillation can be used to provide a suitable interplay between intensification and diversification of the search. Strategic oscillation may be implemented by relaxing a number of problem constraints when applying a given neighbourhood structure. The objective is to increase the number of possibilities to perform a move allowing for the acceptance of non-feasible solutions during a given period of the search. The choice of the number and the type of restrictions which are relaxed depends on the problem. In combinatorial optimisation where problems are frequently defined on graphs, restrictions are imposed on the connectivity of the solution subgraph (of the problem) and on the type of this connectivity (for a particular instance of the problem). For the CSP the connectivity restrictions are satisfied if all the pieces are covered. This type of connectivity imposes restrictions on the sequence of trips grouped together to form pieces and on the pieces matched to form duties.

The Tabu-Crew algorithm is an extension of the run-cutting method described above, implemented in a tabu search framework. Starting with the solution obtained in the construction phase, a new schedule is obtained at each iteration, by removing a certain number of duties from the current solution and running the run-cutting algorithm again to cover the sub-problem.

Since the optimal covering of the timetable is likely to have only a few inefficient duties, with only a single piece of work, any current schedule is likely to improve if the number of those duties is decreased. So, inefficient duties as well as their adjacent duties are removed from the current solution in order to free space for new configurations. Such a procedure, which destroys part of a solution and reconstructs a new solution, passing through a sequence of incomplete (or non-feasible) solutions until all the pieces are covered, represents a form of strategic oscillation.

In addition to strategic oscillation, a diversification strategy was implemented by modifying the evaluation function in order to avoid the insertion of repeated inefficient duties in subsequent iterations. To do so, a long term memory was included in the algorithm. This is a vector that records frequencies of pieces identified by their position in the timetable. The frequency values indicate the number of times that each trip was assigned to a duty with a single piece of work. The evaluation function returns a higher value for duties with two pieces of work that include trips with high frequencies. This is achieved by adding those frequencies to the evaluation function. This strategy enhances the chances that 'difficult trips' will be combined with others, at early stages of the algorithm, avoiding inefficient duties. For a full description of the algorithm see Cavique[16] and Cavique and Themido.[17]

The procedure proved to be very efficient at improving the initial solution after a few iterations, but thereafter a significant amount of time was required to find a better solution. This result is not surprising as the CSP has a very constrained 'puzzle' type structure and to construct a new feasible schedule for a partial timetable it is usually necessary to destroy a relatively large number of pieces. This means that the construction process is expensive from a computational point of view.

Improving the search using ejection chains: the run-ejection algorithm

As described above, when connectivity restrictions are relaxed, oscillation strategies can be obtained by partial destruction of solutions which are progressively reconstructed in a different way using a constructive algorithm. Other processes for relaxing connectivity restrictions are based on ejection chain methods.

Ejection chains incorporate an implicit strategic oscillation produced by a sequence of ejection moves leading to solutions which are partially disconnected or incomplete. However, a complete solution can be obtained by a supplementary move connecting the current solution subgraph. Basically, ejection chain methods work as follows. The neighbourhood of a solution is evaluated by successive transformations of a structure which normally does not represent a complete or feasible solution but serves as reference for the evaluation of trial solutions at each step of the ejection chain. The transformation of one structure into another consists in replacing some components of the current structure by new ones, and the insertion of the new components, forcing other components to leave the structure, defines an *ejection move*. Each structure transformation identifies one *level* of the ejection chain in which a feasible *trial solution* can be obtained by using an appropriate *trial move*. At each level l of the chain we have a *compound move* produced by the cumulative effect of the previous $l - 1$ ejection moves followed by the supplementary trial move of level l. The neighbourhood search usually consists of constructing L levels of one ejection chain from a current solution evaluating the trial solution at each level. Finally, the level l^* at which the best trial solution is found determines the compound move used to transform the current solution into a new one. For a comprehensive description of developments in ejection chain methods and applications we refer the reader to Glover[18] and Rego.[19]

We will describe a subgraph ejection chain method which is incorporated into a tabu search procedure that provides an interesting algorithm for the CSP, exploring the block partition method described below.

Block partition strategy

The block partition strategy is used in some of the most famous CSP packages: in the RUCUS-II and in the initial version of HASTUS.[2] The method is divided into three parts: block partition, graph generation and duty achievement. The method is based on the formulation of the Maximum Cardinality Matching Problem (MCMP) of a non-bipartite graph $G = (P,D)$.

The block partition procedure, in the first step, will divide the blocks into feasible pieces of work creating the node set $P = \{p_1, p_2, \ldots, p_k\}$, (that is the set of all pieces of work). In the second step, the matching graph G is built, linking pairs of pieces for all possible duties, creating the edge set $D = \{(p_i, p_j)/p_i, p_j \in P\}$. Finally, in the third step, the MCMP algorithm runs. The matched nodes are duties with two pieces and the free nodes (or unmatched) are duties with only one piece of work, (see Figure 1).

The difficulty of this approach is that there is a large number of possible graphs G depending on the way that each block $b_l \subseteq P$ is partitioned into pieces. In order to explore this enormous number of alternatives we consider a tabu search framework using compound neighbourhoods defined as follows.

PIECES OF WORK

MATCHING GRAPH

DUTY ACHIEVEMENT
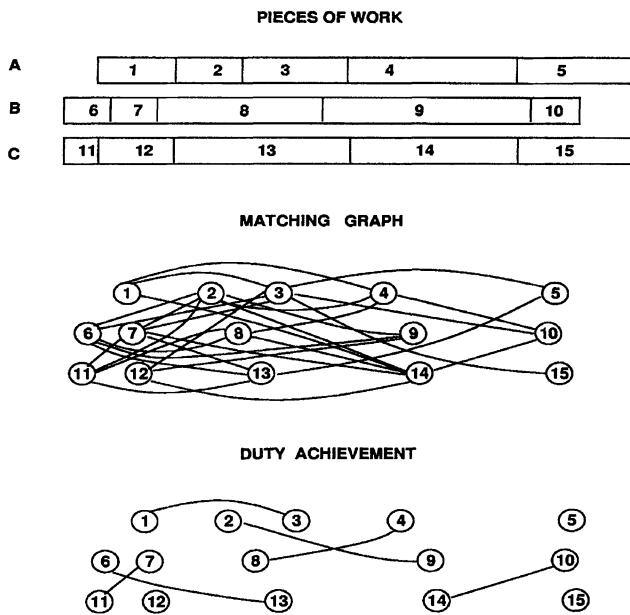
Figure 1 Example of creating duties from a given partition of three into fifteen pieces of work.

Neighbourhood structures

The neighbourhood search is based on an *embedded neighbourhood structure* N consisting of two substructures N_1 and N_2 which separate the neighbourhood space into two subsets. N_1 is an intermediate structure responsible for generating sets of new pieces of work which will replace pieces of the current graph $G_k = (P_k, D_k)$ transforming it into another graph $G_{k+1} = (P_{k+1}, D_{k+1})$.

The final neighbourhood structure N is obtained by choosing a set of edges in G_{k+1} in order to obtain a feasible solution.

Notice that N_1 is an incomplete neighbourhood because it does not define regions of complete (or feasible) solutions but rather changes the neighbourhood space in which a new solution will be searched with N_2. In other words, N_1 defines a set of directions (arcs) on the *solution space graph* to be searched from the current solution S_k. These directions, which restrict the neighbourhood space, are represented by the graph G_{k+1} and a new neighbouring solution S_{k+1} is determined by solving the MCMP on G_{k+1}.

In the algorithm, the neighbourhood structure N_2 is constructed using a greedy heuristic procedure. At each iteration the procedure searches the piece with the smaller number of links (that is the node of G_{k+1} with lower degree) and selects one alternative linking it to another piece.

We define $N = N_1 \oplus N_2$ as the combination of the two neighbourhood structures and use the classical notation, introduced by Hertz and de Werra[20] to denote $S_{k+1} = S_k \oplus m$, the transition from a solution S_k to a new

solution S_{k+1} by the application of a move m. Let X be the set of feasible solutions, M_1 and M_2 the set of moves defined respectively in N_1 and N_2.

By definition, the neighbourhood $N(S_k)$ of a given solution S_k is the set of all S_{k+1} solutions that can be reached from S_k applying the neighbourhood structure N. The neighbourhood considered in the algorithm can be defined as $N(S_k) = \{S_{k+1} | \exists m_1 \in M_1, m_2 \in M_2$ with $S_{k+1} = (S_k \oplus m_1) \oplus m_2, S_{k+1} \in X\}$. In the context of this paper, it will also be useful to consider moves to modify the matching graph rather than solutions of the problem, and we will use the same concepts and notations to treat graphs as those defined for solutions.

We now describe in detail the various components of the neighbourhood structure considered in this tabu search procedure.

A subgraph ejection chain method

As is typical in practical crew scheduling problems, the reduction of the number of duties, even for a randomly generated schedule is hard to realize because problems are very constrained and usually a small change in one part of the schedule requires propagation of changes throughout the entire schedule to produce an improvement.

Generating new pieces of work

Based on this assumption we have conceived an *ejection chain method* in which, at each level l $(l = 1, 2, \ldots, L)$ of the chain, a subgraph G_k^{l-} of G_k^l $(G_k^l = G_k)$ is deleted (ejected) and replaced by another subgraph G_k^{l+} which is added to the current graph transforming G_k^l into $G_k^{l+1} = G_k^l \setminus G_k^{l-} \cup G_k^{l+}$. The process is defined by successive delete/add ejection moves, where each delete operation of a subgraph in order to add a new subgraph leads to the generation of another move of the same type. The process ends when no other pair of these operations is carried out.

In the ejection chain process, delete/add moves can be carried out by any of the elementary operations defined as follows:

- *shift operation*, which shifts the extreme of a piece to the right or to the left transferring one or more trips between adjacent pieces. Given trip number t and two adjacent pieces of work $p_a = \{t, \ldots, t + a\}$ and $p_b = \{t + a + 1, \ldots, t + b\}$ with $b > a + 2$, a right shift returns $p_a^r = \{t, \ldots, t + a + 1\}$ and $p_b^r = \{t + a + 2, \ldots, t + b\}$ and a left shift returns $p_a^l = \{t, \ldots, t + a - 1\}$ and $p_b^l = \{t + a, \ldots, t + b\}$;
- *cut operation*, which cuts one piece into two pieces;
- *merge operation*, combines two pieces into a single piece.

The move m_l mentioned above is composed by combinations of these operations or submoves (one at each level of

the chain), $m_1 = m_{11} \oplus m_{12} \oplus \ldots \oplus m_{1l}$ which defines the ejection chain process generated by N_1. However, such a procedure only defines sequences of *ejection moves* which create sets of new pieces and their corresponding links to existent pieces, modifying G_k at each level of the chain. Figure 2 illustrates an example of three levels of an ejection chain using a sequence of shift, cut and merge operations performed on block b2 for the problem presented in Figure 1.

In the example, pieces 7 and 8 are modified by shifting trips from piece 8 to piece 7 defining the first level of the ejection chain. Subsequently, the matching graph that represents all feasible combinations of two pieces of work, from blocks A, B and C (see Figure 2), that may form a duty must be transformed. Unfeasible edges 7–2 (connecting piece 7 with piece 2) and 7–13, that correspond to overlapping pieces, must be deleted and new edges linking pieces 1–8 and 8–12, which can be matched in this new configuration, must be added. The new edges created by the move are represented by unbroken lines. The dotted edges are those which have been removed from the graph. For the second level of the ejection chain, a cut operation is considered dividing piece 9 into pieces 9 and 16. This operation adds a new node to the matching graph and usually increases the total number of edges of the graph. In the example only two edges, linking node 9 with nodes 2 and 6 are removed (doted lines) while six new edges are added. In contrast, the merge operation used in

Figure 2   Example of three levels of an ejection chain obtained by a combination of shift, cut and merge operations.

the third level of the chain removes edges rather than adding new ones, and reduces the number of nodes. In this latter level, piece 6 is merged with piece 7 which is 'removed' from the graph, leaving it with the same number of pieces as the previous solution (15 pieces). However, the number of duties with a single piece of work was reduced from 3 to 1, thus reducing the total number of duties from 9 to 8. We recall that the final graph is obtained by a sequence of dependent operations, because each operation at a level takes into account the transformations of the graph carried out by operations in previous levels of the chain.

Creating new duties

To obtain a trial solution it is necessary to define a *trial move* to evaluate the complete transition from one solution to another. Now, it is easy to see that a trial move may be done for each level l of the chain by solving the corresponding Maximum Cardinality Matching Problem (MCMP) on the current graph G_k^l. Figure 2 illustrates a trial solution that can be obtained for $l = 3$ in the chain.

Of course, this is extremely expensive from the computational point of view. Therefore, in our implementation the evaluation of the trial move is not explicit. Instead there is a trial function E_k that reflects the performance of the chain for the transition from $G_k^l = (P_k^l, D_k^l)$ to $G_k^{l+1} = (P_k^{l+1}, D_k^{l+1})$. The value of the trial function is expressed in terms of the number of edges and is given by its increase in two successive levels of the chain: $E_k = |D_k^{l+1}| - |D_k^l|$.

The objective is to maximise E_k, the criterion for the selection of a move carried out by one of the three operations defined above. Therefore, the graph of the problem is successively enriched with new edges allowing a greater number of variables (runs) to be evaluated at each iteration k of the algorithm. Once the chain ends, the final move m_2 is carried out by applying N_2 on the graph $G_k^{l^*}$, where l^* represents the level of the chain where the best value of E_k was found. The quality of a solution S is given by the objective function $F(S)$ which measures the corresponding number of duties.

It is clear that when using criterion E_k if no condition is imposed on the selection of operations, the cut operation will be frequently chosen for the move. In order to maintain a good balance of operations and introduce a suitable oscillation of the search, operations are randomly selected throughout the ejection chain.

In the tabu search algorithm, the tabu list is used with two purposes: (1) guiding moves in the ejection chain process; and (2) avoiding the search returning to solutions previously visited. For the first purpose, a tabu tenure of a single iteration is considered, forbidding a piece of work to be modified again, until the end of the ejection chain, in order to propagate the search throughout the entire schedule. This also satisfies the second purpose; nevertheless to
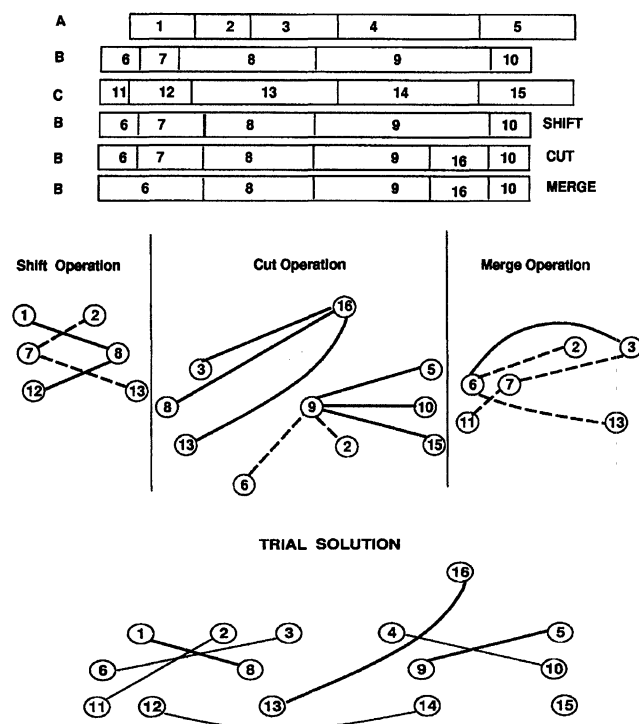
prevent cycling, reverse operations of moves are put into the tabu list for a certain number θ of iterations, which is randomly generated within an interval $[\theta_{\min}, \theta_{\max}]$. Reverse operations are shift-left/shift-right and cut/merge. To implement this, we consider tabu elements based on two attributes of a move, the type of operation, and the piece of work on which the operation is carried out. The size of the tabu list is generally expressed as $\alpha\sqrt{n}$, where α is a parameter bounded by $0 \leqslant \alpha \leqslant 1$ and n is the number of elements in the problem that can be tenured. In the LU case $\theta_{\min} = 0.5\sqrt{|P|}$ and $\theta_{\max} = \sqrt{|P|}$ where $|P|$ represents the number of pieces of work.

The Subgraph Ejection Chain procedure can thus be described as follows, we denote:

—L the maximum number of levels of an ejection chain;
—E_k the value of the trial move in iteration k;
—$G^l = (P^l, D^l)$ the matching graph at level l of the ejection chain.
—$F(S_k)$ the cost of solution S in iteration k,
—S^* the best solution found so far, initially the solution produced by the run-cutting algorithm.
—K the maximum number of iterations without any improvement;
—TL the tabu list;
—$\theta_{\min}, \theta_{\max}$ bounds for the tabu tenure.

Procedure: *Subgraph Ejection Chain* $(S^*, L, \theta_{\min}, \theta_{\max}, K)$
   Set $k = 0$;
   Set $TL = \emptyset$.
   Construct the initial matching graph G_k^0 from the set of pieces P in S^*. Perform while $k < K$
• Initialisation:
   Set $k = k + 1$ and set $l = 0$.
   $E^* = E_k = -|D_k^l|$.
• Perform the ejection chain:
   For each $l = 1$ to L do
     • For each piece $p_i \in P^l$ select at random one of the three move operations for m_{11}.
     • Select $m_{11} \notin TL$ that maximises $E_k = |D_k^{l+1}| - |D_k^l|$.
     • Perform the transition from G_k^l to $G_k^{l+1} = G_k^l \oplus m_{11} = G_k^l \backslash G_k^{l-} \cup G_k^{l+}$.
     • Put attributes of m_{11} into the tabu list TL with a tabu tenure of one iteration.
        If $E_k > E^*$, set $E^* = E_k$, set $l^* = l$.
• Evaluate the neighbouring solution S_k:
   Generate a tabu tenure θ in $[\theta_{\min}, \theta_{\max}]$ and put attributes of the ejection moves $m_{1l}, \ldots, m_{1l}$ into the tabu list for the next θ iterations.
   Construct the matching graph $G_k^{l^*}$.
   Perform m_2 by solving the MCMP on $G_k^{l^*}$ obtaining S_k.
   If $F(S_k) < F(S^*)$ set $S^* = S_k$; set $k = 0$.
   Set $G_k^0 = G_k^{l^*}$.
Return S^*.

In addition to the strategic oscillation implicit in the ejection chain process, a diversification strategy was implemented by introducing an additional term in the move evaluation function in order to favour the insertion of edges linking nodes (pieces) which have frequently represented duties with a single piece of work. To do so, we consider a vector to record frequencies of pieces identified by their positions. The frequency values indicate the number of times that each piece was left isolated in the solution of a matching problem.

Computational results

The performance of the Tabu-Crew (TC) and Run-Ejecting (RE) algorithms was evaluated for a set of six timetables. Results are also presented for the Run-Cutting (RC) algorithm, which provides the initial schedule for the TC and RE algorithms. Comparisons were made between the results obtained by our algorithms and manual procedures of Lisbon Underground (LU).

The computational results for the algorithms are reported in Table 1. The computations were performed on a HP 9000/712 workstation and algorithms were coded in Pascal. The quality of the solutions is evaluated on the basis of three correlated performance measures: the percent improvements to the number of duties obtained by LU manual schedulers; the matching ratio, that is the percentage of duties with two pieces of work; and the average number of driving hours per duty. The LU considers that any schedule with an average of 4.5 driving hours per duty is of very good quality.

The Run-Cutting algorithm does not provide solutions superior to those obtained manually by LU but seems to be a fast algorithm, good enough to produce initial solutions. Both Tabu-Crew (TC) and Run-Ejecting (RE) algorithms reduced the number of duties in the initial schedule. This improvement represents a reduction of 1.8% and 3.6% on the manual solutions produced by the LU planners. However, the Run-Ejecting algorithm produces better quality duties than Tabu-Crew and achieves the solutions in a much shorter computational time which indicates a superior algorithm.

Improvements for the old timetables (U62, U63 and U65) are also smaller than those obtained for the more recent ones. This is natural as the manual solutions for crew scheduling, which were in operation over long periods of time, were improved by experience and by trial and error, while for the new crew schedules no such learning possibility exists. This shows the importance of the algorithms for the future operation of the LU as many alterations to the track layout are considered for the near future.

We have verified that it is hard to find feasible solutions for timetables with small blocks (one or two pieces of work) as occurs in the weekday timetables U62 and U68

Table 1 Performance measures and computational times

	Number of duties and driving time per duty (h)								% Improvement on LU solution			% Matching			CPU time (s)		
	LU		RC		TC		RE		RC	TC	RE	RC	TC	RE	RC	TC	RE
U62	76	4.5	79	4.3	76	4.5	75	4.5	−3.9	0.0	1.3	59	63	65	1.7	1642	134
U63	66	4.6	68	4.5	65	4.7	64	4.8	−3.0	1.5	3.0	70	72	75	1.2	639	87
U65	80	4.4	83	4.2	80	4.4	78	4.5	−3.8	0.0	2.5	63	66	69	1.3	863	92
U68	90	4.1	92	4.0	87	4.2	86	4.3	−2.2	3.3	4.4	66	70	71	1.9	1254	153
U65a	80	4.4	82	4.3	79	4.4	76	4.6	−2.5	1.3	5.0	71	75	77	1.2	298	97
U68a	90	4.1	91	4.0	86	4.3	85	4.3	−1.1	4.4	5.6	69	74	76	1.5	743	85
Avg	80	4.3	83	4.2	79	4.4	77	4.5	−2.8	1.8	3.6	66	70	72	1.5	907	108

that include a number of trains with broken blocks, resulting from trains being withdrawn from the track during off-peak periods. This may justify the longer computational times and suggests that computational time depends more on the size of blocks than on the number of trips.

Conclusions

Two heuristic algorithms for a real crew scheduling problem faced by the Lisbon Underground were presented. The proposed algorithms are part of a decision support system for crew scheduling management. The objective is to create independent modules, which can be successively integrated in a complex system.

In a first stage an efficient run-cutting algorithm, which rapidly provides an initial feasible solution was developed. The algorithm was adapted to consider local improvements of the initial schedule using a tabu search framework. The implementation of the tabu-crew algorithm has shown the advantage of using strategic oscillation for searching new crew scheduling solutions.[16,17]

In a second stage we have developed another algorithm, based on the block partition strategy, in order to search for better scheduling configurations in a wider solution space. Ejection chains have produced interesting results in a number of scheduling related problems. Motivated by successful experiments obtained from node based ejection chains[21] and subpath ejection chain methods,[22,23] a new type of ejection chain based on the ejection of subgraphs was introduced for the crew scheduling problem. Once again, the method proved quite effective for finding good solutions quickly.

It is worth noticing that both algorithms used an oscillation strategy: the tabu-crew did so in an explicit way and the run-ejecting in an implicit way. The ejection chains incorporate an implicit strategic oscillation produced by a sequence of ejection moves leading to solutions, which are partially disconnected or incomplete. Both approaches ensure that a feasible solution is obtained in a finite number of steps. In the first approach, this value is bounded by the number of iterations of the constructive algorithm, whereas in the second approach a feasible solution can be obtained at each step of the ejection chain process.

References

1 Bodin L and Berman D (eds) (1975). *Workshop on Automated Techniques for Scheduling of Vehicles Operator for Urban Transportation Services*, Pre-print.

2 Wren A (ed) (1981). *Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling*. North-Holland, Amsterdam.

3 Rousseau JM (ed) (1985). *Computer Scheduling of Public Transport 2*. North-Holland, Amsterdam.

4 Daduna J and Wren A (eds) (1988). *Proceedings of the Fourth International Workshop on Computer-Aided Scheduling of Public Transport*. Lecture Notes in Economics and Mathematical Systems, Springer-Verlag, Berlin, Heidelberg.

5 Desrochers M and Rousseau JM (eds) (1992). *Proceedings of the Fifth International Workshop on Computer-Aided Scheduling of Public Transport*. Lecture Notes in Economics and Mathematical Systems, Springer-Verlag, Berlin, Heidelberg.

6 Daduna J, Branco I and Paixão J (eds) (1995). *Proceedings of the Sixth International Workshop on Computer-Aided Scheduling of Public Transport*. Lecture Notes in Economics and Mathematical Systems, Springer-Verlag, Berlin, Heidelberg.

7 Wilson N (ed) (1997). *Proceedings of the Seventh International Workshop on Computer-Aided Scheduling of Public Transport, 1997*. To appear in Lecture Notes in Economics and Mathematical Systems, Springer-Verlag, Berlin, Heidelberg.

8 Bodin L, Golden B, Assad A and Ball M (1983). Routing and scheduling of vehicles and crews—The state of the art. *Comp and Opns Res* **10**: 63–212.

9 Wren A and Rousseau J (1995). Bus driver scheduling: an overview. In: Daduna J, Branco I and Paixão J (eds). *Proceedings of the Sixth International Workshop on Computer-Aided Scheduling of Public Transport*. Lecture Notes in Economics and Mathematical Systems, Springer-Verlag, Berlin, Heidelberg, pp 173–187.

10 Hartley T (1981). A glossary of terms in bus and crew scheduling. In: Wren A (ed). *Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling*, North-Holland, Amsterdam, pp 353–359.

11 Falkner J and Ryan D (1992). Express: Set partitioning for bus crew scheduling in Christchurch. In: Desrochers M and Rousseau JM (eds) (1992). *Proceedings of the Fifth International Workshop on Computer-Aided Scheduling of Public Transport.* Lecture Notes in Economics and Mathematical Systems, Springer-Verlag, Berlin, Heidelberg, pp 359–378.

12 Rich E and Knight K (1991). *Artificial Intelligence.* McGraw-Hill, USA.

13 Reingold E, Nievergelt J and Deo J (1977). *Combinatorial Algorithms.* Prentice-Hall, New Jersey.

14 Glover F (1989). Tabu search—Part I. *ORSA J Comp* **1**: 190–206.

15 Glover F and Laguna M (1997). *Tabu Search.* Kluwer Academic Publishers, Boston, MA.

16 Cavique L (1994). Sequenciamento de pessoal tripulante. M.S. Thesis, Universidade Tecnica de Lisboa—Instituto Superior Tecnico, Portugal.

17 Cavique L and Themido I (1995). Sequenciamento de servicos de pessoal tripulante: uma abordagem baseada num conjunto de heuristicas. *Invest Opl* **15**: 123–141.

18 Glover F (1992). Ejection Chains, reference structures and alternating path methods for the travelling salesman problem. Technical Report, Graduate School of Business and Administration, University of Colorado at Boulder, USA.

19 Rego C (1996). Local Search and Neighborhood Structures for Vehicle Routing Problems: Sequential and Parallel Algorithms. PhD Thesis, PRiSM Laboratory, University of Versailles, France.

20 Hertz A and de Werra D (1987). Using tabu search techniques for graph coloring. *Computing* **39**: 345–351.

21 Rego C and Roucairol C (1996). Parallel tabu search algorithm based on ejection chains for the vehicle routing problem. In: Osman IH, Kelly JP (eds). *Metaheuristics: Theory and Applications.* Kluwer Academic Publishers, Boston, MA.

22 Rego C (1998). Relaxed tours and path ejection for the traveling salesman problem. *Eur J Opl Res* **106**: 522–538.

23 Rego C (1998). A subpath ejection method for the vehicle routing problem. *Mmt Sci* **44**: 1447–1459.