Examination Timetables and Tabu Search with Longer-Term Memory

George M. White and Bill S. Xie

School of Information Technology and Engineering,
University of Ottawa,
Ottawa, K1N 6N5, Canada
white@site.uottawa.ca

Abstract. The examination scheduling problem has been examined and a four-phase system using a tabu search algorithm, OTTABU, has been implemented. This system uses both recency-based short-term memory and move (or frequency)-based longer-term memory to improve the quality of the solutions found. The system was tested using real data obtained from the University of Ottawa registrar's office and real examination schedules were produced. It was found that the use of longer-term memory produced better schedules that those produced without such memory – typically a 34% improvement was obtained due to this factor alone. The length of the long term memory list was also found to be important. A length that is too small can greatly reduce its effectiveness. A list that is too long only reduces the effectiveness by a small amount. A quantitative analysis method is applied to estimate the appropriate length of the longer-term tabu list and a controlled tabu relaxation technique is used to improve the effectiveness.

1 Introduction

The examination timetabling problem is a difficult combinatorial exercise that has been studied for many years, dating back to at least the early 1960s [1]. This work has led to several implementations that have been used with much success at universities [2], [3].

A review of most of the work published in this area can be found in the comprehensive coverage of Carter and Laporte [4].

Much of the early work was based on bin packing algorithms with heuristic rearrangement of the schedule based on some version of the travelling salesman's algorithm to reduce the number of consecutive examinations taken by students. More recent work is based on the observation that the examination timetabling problem is an assignment-type problem and can be considered to be a graph colouring problem. Some researchers have applied tabu search (TS) techniques to solving examination timetabling problems in recent years. Hertz and de Werra [5] used a tabu search algorithm to generate solutions for the problem modelled as a graph colouring problem with great success. After this initial approach, Hertz [6] applied the approach developed for graph colouring to construct and

E. Burke and W. Erben (Eds.): PATAT 2000, LNCS 2079, pp. 85–103, 2001.

solve course timetabling and examination timetabling problems. Boufflet et al. [7] modified this method to solve a particular practical examination timetabling problem. They found that their TS techniques improved the solutions well.

In these works only recency-based short-term tabu techniques were used. The move *(exam, original period)* is stored instead of the solution into the tabu list in order to save space and time, and the tabu tenure is set to a fixed value 7. Boufflet et al. also set *nbmax*, the maximum number of null iterations, to 200 but Hertz did not indicate which value he used. Hertz generated the neighbourhood of a current solution s only from the exams which create at least one conflict in s, but Boufflet et al. considered all the feasible moves of exams from one feasible set to another one.

In this paper, we describe an automated TS approach such that a frequency-based longer-term memory mechanism combined with tabu relaxation technique is used to optimize an examination timetable problem. The tabu relaxation technique can accelerate downhill movement and diversify the search space. We also introduce a quantitative analysis method which investigates the examination distribution and estimates the size of the lightest or the most active exams and the size of the heaviest or the most inactive exams in the examination set, and makes it possible to choose automatically the appropriate parameters for TS for an individual examination timetabling problem.

Section 2 describes the examination timetabling problem formulated as a graph colouring problem. The following sections describe the ideas behind TS and show how a specific implementation of a TS algorithm can be used to cast timetables with certain desirable qualities. Data taken from the Fall 1996 registration data at the University of Ottawa were used to test the system using real data. Some comparisons with results obtained from other researchers conclude this paper.

2 Graph Colouring and Examination Timetabling Models

The examination timetabling problem is an assignment-type problem and can also be considered to be a graph colouring problem. The assignment-type problem can be described as follows [8]:

> Given n items (exams) and m resources (timeslots), determine an assignment of each item to a resource in order to optimize an objective function and satisfy s additional side constraints (classified as hard and soft constraints).

We use an undirected graph $G = G(V, E)$ to describe the examinations and their relationships. Let V be the examination (or node) set, v be the number of examinations, v_i be the ith examination and s_i be the number of students taking exam v_i. If there are m students who must take both exam v_i and v_j, we consider there is an edge e_{ij} with a weight m between the node v_i and v_j. Let E be the edge set in the graph, e the total number of edges in the graph and w the sum of the weights of edges in E.

Let T be a given set of consecutive timeslots, each one containing zero or more exams: i.e. $T = (T_1, T_2, \ldots, T_k)$, where k is the number of timeslots. C_i is the maximum number of seats available for T_i. Each exam in the graph is assigned into a timeslot T_i. For a timetable to be *feasible*, the sum of weights of edges having both endpoints in the same timeslot must be zero. In addition, the sum of weights of edges between the timeslots with a given distance (i.e. of adjacent timeslots) should be minimized. Such a timetable is called optimal. We must, of course, ensure that no timeslot requires more examination seats than are available.

Let $E_{i,0}$ be the set of edges having both endpoints in T_i $(i = 1, 2, \ldots, k)$, $E_{i,1}$ be the set of edges between T_i and T_{i+1} $(i = 1, 2, \ldots, k-1)$ and $E_{i,2}$ be the set of edges between T_i and T_{i+2} $(i = 1, 2, \ldots, k-2)$. Let W_0 be the sum of weights of edge set $E_0 = \cup E_{i,0}$ $(i = 1, 2, \ldots, k)$. W_1 is the sum of weights of edge set $E_1 = \cup E_{i,1}$ $(i = 1, 2, \ldots, k-1)$. W_2 is the sum of weights of the set $E_2 = \cup E_{i,2}$ $(i = 1, 2, \ldots, k-2)$. We define the objective function $f(s)$ for solution s as

$$f(s) = p_0 \times W_0 + p_1 \times W_1 + p_2 \times W_2$$

where s is subject to the condition that for all T_i, the sum of the enrolments $\leq C_i$ (the maximum number of seats available for T_i). p_0, p_1 and p_2 are the penalties chosen to weight the different conflict classes in s that are characterized by W_0, W_1 and W_2 respectively. For this problem we set $p_0 \gg p_1 \gg p_2 = 1$.

Hence, the problem of optimal examination scheduling is equivalent to minimizing the value of the objective function $f(s)$ of a solution set S.

3 Concepts of Tabu Search

The TS optimization schemes are derived from ideas proposed at various times in the 1960s by Fred Glover, and subsequently developed by him and by other researchers [9], [10]. As defined in [10],

> Tabu search is a meta-heuristic that guides a local heuristic search procedure to explore the solution space beyond local optimality.... The local procedure is a search that uses an operation called *move* to define the neighbourhood of any given solution.
>
> A meta-heuristic refers to a master strategy that guides and modifies other heuristics to produce solutions beyond those that are normally generated in a quest for local optimality. The heuristics guided by such a meta-strategy may be high level procedures or may embody nothing more than a description of available moves for transforming one solution into another, together with an associated evaluation rule.

The local search strategies are often memoryless ones that keep no record of their past moves or solutions. The TS meta-heuristic makes abundant use of memory in two ways. Different *adaptive memory structures* are often incorporated that remember some of the recent moves made by the algorithm and may record some

of the more recent or more promising solutions found. This memory of the search history is used to incorporate a *responsive exploration* of the state space helping the search to find superior solutions faster.

Glover and Laguna [10] emphasize four important dimensions of the memory structures used in TS. These are

- *recency*: The TS memory keeps track of the solution attributes that have changed in the recent past. These attributes are parts of the solution that change in moving from one solution to another.
- *frequency*: TS calculates ratios that keep track of which attributes change the most and how often they move.
- *quality*: In principle, TS can distinguish a better solution from a worse one by using criteria other than a single objective function, i.e. TS can directly incorporate multiple-criteria decision capability.
- *influence*: TS uses the information it has in its memory to evaluate the choices it will make and the quality of the solutions it finds.

The strategies used by TS can be classified as using *intensification* and/or *diversification* [10]. The strategy of intensification is implemented by modifying the choice rules used to select moves in order to aid the location of solutions that have been found to be good in previous search areas. The basic idea here is that if certain regions have yielded good solutions in the past, they may well yield even better solutions in the future.

When diversification is used, the exact opposite behaviour is encouraged. This drives the search area into those regions that have not yet been well explored. Perhaps better solutions can be found there, as that area has not yet been well examined.

4 The OTTABU Algorithm

We have implemented a TS algorithm OTTABU and have used it in an attempt to provide a practical examination timetable using data provided by the University of Ottawa. The data were generated in the Fall of 1996 and are based on the course enrolments of that time. This section outlines the strategies used in developing the algorithm. Details are given in Section 6.

4.1 Initial Solution

An algorithm derived from a bin packing algorithm (largest enrolment first) is used to generate an initial solution. The main idea for this algorithm is presented in the next paragraph. Note that this initial solution may be either feasible or infeasible.

Let G be a set of examinations and T a set of timeslots. G consists of v exams with enrolments $s_1, s_2, \ldots, s_v$ and T consists of t timeslots with seating capacities $C_1, C_2, \ldots, C_t$. We assign the exam with the largest enrolment from G, denoted as A, into the first timeslot T_1. Then we construct a set of examinations

G^A (a subset of G) whose members have no conflicts with the exam A, and try to schedule the largest exam in G^A into T_1. If T_1 does not have the capacity to accept it, try the second largest one.... Repeat this procedure until T_1 is full or no exam without conflict with exams assigned in T_1 is available. This procedure can guarantee timeslot T_1 to be conflict free. Apply the same algorithm to the unassigned exams for $T_2, T_3, \ldots, T_t$. If one or more of the timeslots is empty and all exams are assigned, we have a feasible timetable. If all t timeslots are used up and there are still some exams to be assigned, we assign them into these non-full timeslots regardless of the conflicts generated.

4.2 Atomic Move, Neighbourhood, and Local Search

Let a solution $s = (T_1, T_2, \ldots, T_t)$, where T_i is the set of exams assigned to timeslot T_i and t is the number of timeslots. We generate a new solution s' from a solution s by an *atomic move*. An atomic move is one such that exactly one node x in s is moved from a timeslot T_i to another timeslot T_j, denoted as (x, i, j). We call s' a neighbour of s and all the neighbours generated from s by an atomic move as the *neighbourhood* of s. The size of the neighbourhood depends on the size of candidate node lists and the number of timeslots. There are two useful types of candidate node lists: those composed of all nodes in the graph and those containing only those nodes related to the conflicts concerned [7]. These are used to generate the neighbours of the current solution. Obviously, the former is V, and the latter is the set of nodes, denoted by V^*, which contains the end-nodes of edges in E_0, E_1 and E_2.

Starting from an initial solution, the TS algorithm iteratively explores a subset $N^*(s)$ of the neighbourhood $N(s)$ of the current solution s. The member of $N^*(s)$ that gives the minimum value of the objective function becomes the new current solution independently of the whether its value is better or worse than the value corresponding to s (i.e. an uphill move is acceptable). If $N(s)$ is not too large, it is possible and convenient to take $N^*(s) = N(s)$.

4.3 Recency-Based Short-Term Memory

Whenever a node x is moved from timeslot T_i to T_j, a move denoted by (x, i, j), to get a new current solution s^*, (x, i) becomes *tabu* and is put into a tabu list TS with a given tenure (so-called short-term memory). The tenure of each entry already in the tabu list is decreased by 1 and those entries with zero tenure are dropped from the tabu list. We choose 9 as the maximum tenure for TS. If a move (x, i, j) creates the best solution so far, we will accept this move regardless of its tabu status in TS, and if (x, j) is in TS, (x, j) will be dropped from TS. We have found that if both long-term and short-term memory are used, the tenure of the short-term tabu list is not critical.

4.4 Transitional Frequency-Based Longer-Term Memory

In order to investigate the effect of a frequency-based long-term memory, we have incorporated a move frequency table (MFT) to store the move frequency of each

node in the graph. When a node is moved, its move frequency is incremented by 1. We use a second, longer-term tabu list TL to dynamically forbid moving over-active nodes in order to get diversification and help to prevent cycling. If a node x has been moved more than two times and TL is not full, it will be put into TL. If it is full and if some node y already in TL has a lower move frequency than x, we drop y from TL and add x into TL. Hence, a node in TL will not be dropped unless TL is full and a new node with higher move frequency is added. If a node is in TL and if we move this node to get a better solution than the latest best solution, we will accept this move but will not drop this node from TL.

In order to choose an appropriate parameter for the length of TL, we need to analyse the properties of the examination set to estimate the size of the lightest and the heaviest exams in the graph. We call a node (exam) with lower enrolment, degree or weight a *light* node (exam). Similarly, a node (exam) with higher enrolment, degree or weight is a *heavy* node (exam).

From the point of view of local search (move), the lightest exams usually are the most active, and the heaviest are the most inactive and most influential if moved. In the process of search, the light nodes can act as "crack fillers" [10] to perform a fine-tuning function. This, of course, helps the generation of more optimal solutions. But, if there are too many light nodes in the graph, there will be a high likelihood that the light exams may repeatedly move from one timeslot to another timeslot. The values of the objective function of the solutions examined will then change by only small amounts or have no change at all. This increases the likelihood of cycling in the TS. The cycling may waste time and iterations, lure the search away from the optimal region, and cause the search to stop prematurely.

Hence, estimating the number of the potentially active nodes will help us to decide what is the appropriate length for TL, denoted as l_{TL}.

4.5 Tabu Relaxation

Another strategy used is the relaxation of tabu lists. If a given number of iterations (this number should be less than $nbmax$) has elapsed and TL is full since the last best solution was found, or if the current solution is much worse than the last best solution, we empty all entries in both TS and TL. Relaxation of the tabu lists will change the neighbourhood of the current solution dramatically, which may drive the search into a new region and increase the likelihood of finding a better solution.

In our algorithm, the maximum number of null iterations $nbmax$ is set at $3 \times l_{TL}$. If the search has passed $2 \times l_{TL}$ iterations since the latest best solution was found and TL is full, or if the value of the current solution is better than the value of the latest best solution by a predefined threshold r, the entries in TS and TL will be dropped automatically. That is, if $(f(s) - f(best))/f(best) > r$, where $f(s)$ is the value of objective function of the current solution s and $f(best)$ is the value of objective function of the latest best solution, both lists are emptied.

Experiments show that the threshold r should be set between 5% to 15%. It is suggested that at the early stage of the search, r should be set to the high end and gradually decreased as better and better solutions are uncovered. This is because it is found that a threshold r that is too high may lead to too much uphill movement such that the search may not be able to go downhill to the best solution after relaxation of tabu lists

It is found that if the number of iterations has passed from l_{TL} to $2 \times l_{TL}$ and TL is full, the relaxation will lead to a dramatic downhill movement, which may in turn lead to new regions or search spaces.

4.6 Intensification

For a large-scale examination timetable, the neighbourhood $N(s)$ of the current solution s generated with nodes in V has a much larger size than the neighbourhood $N^*(s)$ generated with the nodes in V^*. We also noticed that the V^* becomes smaller and smaller while the solution is getting better and better. Hence, we use a strategy such that a smaller *nbmax* is chosen for the search over V and a larger *nbmax* chosen for the search over V^*. It is shown that the four-pass search strategy presented below can generate better solutions and save search time.

Pass 1: We start the TS from a given initial solution. We generate neighbours of current solution s with the nodes in V^* and set a bigger value for *nbmax* to allow the search to do more downhill–uphill movements which may increase the likelihood that more optimal solutions may be found, We store the best solution and the last solution. The last solution may be as good as, or worse than, the best solution.

Pass 2: We use the last solution obtained in pass 1 as the initial solution to start a new search round with smaller l_{TL} and *nbmax* (compared with the size of V). The best solution obtained in pass 1 is saved as the default best solution. We erase both the short-term memory and longer-term memory before starting the search. The neighbourhood of the current solution is generated from V. We have found that this will change the search behaviour and solution space dramatically and not take much more time because the *nbmax* is small.

Pass 3: We repeat pass 1 with the last solution and the best solution obtained in pass 2.

Pass 4: We repeat pass 2 with the last solution and the best solution obtained in pass 3. The best solution obtained in this pass is the final solution.

Obviously, this strategy is an intensification strategy in which each pass except Pass 1 intensifies the search in the region containing the best solution obtained in the previous pass. We found that this strategy works well.

5 Quantitative Analysis of Examination Graph

In order to choose an appropriate parameter for the length of TL, we need to analyse the graph of the examination set. We define the *degree* d_i of a node v_i

to be the number of edges connected with v_i directly, and the sum of weights of these edges w_i to be the *weight* of the node.

We calculate:

- the total number of edges, $e = \sum d_i/2 \qquad (i = 1, 2, \ldots, v)$
- the total enrolment, $s = \sum s_i \qquad (i = 1, 2, \ldots, v)$
- the total weights of graph, $w = \sum w_i/2 \qquad (i = 1, 2, \ldots, v)$
- the density of matrix, $dom = 2e/v(v-1)$
- the average enrolment per exam, $\bar{e} = s/v$
- the average degree per exam, $\bar{d} = 2e/v$
- the average weight per exam, $\bar{w} = 2w/v$.

Experiments have shown that the nodes (exams) with lower enrolment, degree or weight have higher probability to be the most active nodes. This, of course, not only helps the generation of more optimal solutions but also increases the probability of cycling in the TS. Hence, estimating the number of the potentially active nodes may help to decide what is the appropriate length for the longer-term tabu list TL.

If the nodes $(v_1, v_2, \ldots, v_v)$ are sorted according to the number of students enrolled, by the degree of node, and by weight of node, we can create three ordered sets:

$$\begin{aligned} s' &= (s'_1, s'_2, \ldots, s'_v) \qquad \text{where } s'_i \le s'_j \text{ for any } i < j, \\ d' &= (d'_1, d'_2, \ldots, d'_v) \qquad \text{where } d'_i \le d'_j \text{ for any } i < j, \\ w' &= (w'_1, w'_2, \ldots, w'_v) \qquad \text{where } w'_i \le w'_j \text{ for any } i < j. \end{aligned}$$

Based on s', d' and w', we estimate how many light and heavy nodes there are in the graph by two methods. The first method is called mean points estimation and the second method is called accumulation percentage estimation.

5.1 Method 1: Mean Points Estimation

For a series $z = z_1, z_2, \ldots, z_m$, sorted in ascending order, we calculate the mean value of z, $\bar{z} = \sum z_i/m$. If the kth element z_k is the largest element $le\bar{z}$ then there are k elements with a value less than or equal to $\bar{z}$. We calculate five points $\bar{z}/3,\ \bar{z}/2,\ \bar{z},\ 2\bar{z}$ and $3\bar{z}$, and find the corresponding k. We apply this procedure to series $s',\ d'$ and w' to get Table 1.

The numbers in the table are the number of nodes with values (enrolment, edge, weight) less than or equal to the corresponding mean points. For example, the value 311 means that there are 311 exams for which the enrolment is less than 1/2 the average enrolment. We define nm to be the average of the number of exams below the mean point of the three series $s',\ d'$ and w'. Thus, the entry $300 = (311 + 253 + 336)/3$. The number of nodes in the graph $v = 771$.

From this table, we estimate there are about 194 light exams (below the mean point $\bar{z}/3$) and 32 heavy exams (above the mean point $3\bar{z}$).

Table 1. Mean Points (data: University of Ottawa, 1996)

Mean point	s'	d'	w'	nm	nm/v
min	1	10	10	7	
$\bar{z}/3$	211	143	229	194	0.25
$\bar{z}/2$	311	253	336	300	0.38
$\bar{z}$	533	488	533	518	
$2\bar{z}$	681	687	680	683	
$3\bar{z}$	734	753	731	739	
max	771	771	771	771	

5.2 Method 2: Accumulation Percentages Estimation

From s', d' and w', we calculate ss, dd and ww respectively to get three new series:

$$ss = s'_1, s'_1 + s'_2, s'_1 + s'_2 + s'_3, \ldots, \sum s'_i \qquad (i = 1, 2, \ldots, v)$$

i.e. $ss = (ss_1, ss_2, \ldots, ss_v)$ where $ss_j = \sum s'_i$ $(i = 1, 2, \ldots, j)$, for $j = 1, 2, \ldots, v$, which indicates the accumulative enrolments of the j lightest exams ranked by their enrolment;

$$dd = d'_1, d'_1 + d'_2, d'_1 + d'_2 + d'_3, \ldots, \sum d'_i \qquad (i = 1, 2, \ldots, v)$$

i.e. $dd = (dd_1, dd_2, \ldots, dd_v)$ where $dd_j = \sum d'_i$ $(i = 1, 2, \ldots, j)$, for $j = 1, 2, \ldots, v$ which indicates the accumulative degrees of the j lightest exams ranked according to their degree;

$$ww = w'_1, w'_1 + w'_2, w'_1 + w'_2 + w'_3, \ldots, \sum w'_i \qquad (i = 1, 2, \ldots, v)$$

i.e. $ww = (ww_1, ww_2, \ldots, ww_v)$ where $ww_j = \sum w'_i$ $(i = 1, 2, \ldots, j)$, for $j = 1, 2, \ldots, v$, which indicates the accumulative weights of the j lightest exams ranked according to their weight.

Obviously, we have $s = ss_v$ (the total enrolment), $e = dd_v/2$ (the number of edges), and $w = ww_v/2$. ss_i/ss_v is the ratio of the accumulative enrolment of the i lightest nodes to the total enrolment. For the same reason, dd_i/dd_v (or ww_i/ww_v) are respectively the ratio of the accumulative degrees (or weights) of the i lightest nodes to $2e$ (or $2w$). We choose the percentages 5%, 10%, 15%, 50%, 85%, 90% and 95% and find the corresponding numbers k in the series s', d' and w'.

We define np to be the average of the number of exams below the percentages of the three series.

Note that 196/766 or 25% of the total nodes in the graph have only 5% of the total enrolment, degree, and weights. It is not difficult to conclude that the lowest 25% of the nodes (196 nodes) in the graph have a greater likelihood to be moved in the process of local search.

Table 2. Accumulation percentages (data: University of Ottawa, 1996)

Percentage	s'	d'	w'	np	np/v
5%	199	171	218	196	0.25
10%	299	258	316	291	0.38
15%	372	321	388	360	0.47
50%	648	597	653	631	
85%	757	738	757	751	
90%	763	752	763	759	
95%	768	763	768	766	

5.3 Estimating the Length of the Longer-Term Tabu List

We use the data calculated above to estimate the number of the lightest nodes in the graph. In general, the nodes located in the area $A = (0, \bar{z}/3) \cup (0, 5\%)$ are the most active nodes. The length of the long-term tabu list l_{TL} can be set to the number of elements in area A: for instance, the largest one among np or nm.

If A is too small for some individual application, we can choose area $B = (0, \bar{z}/2) \cup (0, 5\%)$ or something similar.

6 Details of the OTTABU Algorithm

Procedures

/ * *Initialization* * /
set np by analysing the examination set G with accumulation percentage method;
s := BinPackingWithLargestEnrolmentFirst(G);
set p_0, p_1, and p_2;
l_{TS} := 9; nbiter := 0; bestiter := 0; optiter := 0; bestsol := s;
fmin := 0; NeighbourType := V^*;

/ * *A Four Pass Tabu Search* * /
t := 0;
while t < 4 do {
ExamTimeTableTabuSearch();
if NeighbourType = V^* then NeighbourType := V
else NeighbourType := V^*;
t := t + 1;
} endwhile;
return;

/ * *ExamTimeTableTabuSearch* * /
MFT := ∅; TS := ∅ ; TL := ∅; nv := np; nbmax := 0;
while $f(bestsol) > fmin$ and $(nbiter - bestiter) < nbmax$ do {
nbiter := nbiter + 1; optiter := optiter + 1;

if NeighbourType = V^* then VC := V^* else VC := V;
if |VC| ≤ nv then nv := 2|VC|/3;
if NeighbourType = V^* then { l_{TL} := nv; nbmax := $3l_{TL}$; }
else { l_{TL} := nv/3; nbmax := $3l_{TL}/2$;}
For any $x \leq VC$, generate NS(s) := $\{s_i | s_i := s \oplus m_i\}$ from all atomic moves $m_i := (x, T_i, T_j)$, where either ((x, j) not in TS) ∧ (x not in TL)) or $f(s_i) < f(bestsol)$;
Choose a solution $s^* \in NS(s)$ with minimum f over NS(s) and if more than one solution is minimal, choose one with lowest move frequency;
$s := s^*$;
update MFT(x); update TS(x, i, j); update TL(x);
if $f(s) < f(bestsol)$ then { bestsol = s; bestiter := nbiter; optiter := 0}
elseif optiter > $2l_{TL}$ or $(f(s) - f(bestsol))/f(bestsol) > r$
then { optiter := 0; TS := ∅ ; TL := ∅;}
} endwhile;
return;

Notes

If, in some iteration, we get a new current solution $s^* = s \oplus (x, T_i, T_j)$, then we do

1. Update $TS(x, i, j)$: All tenures are decreased by 1. Drop the entries whose tenures are 0, and add (x, i). If s^* is the best solution so far, drop (x, j) if it exists;
2. Update $TL(x)$: add (x) into TL if TL is not full and $MFT(x) \geq 2$, or replace (y) in TL with (x) if TL is full and $MFT[y] < MFT[x]$;
3. Update $MFT(x)$: $MFT[x] := MFT[x] + 1$.

6.1 Symbols

TS, l_{TS}, a short-term tabu list and its maximum tenure;
TL, l_{TL}, a longer-term tabu list and its maximum length;
np, the estimated number of most active nodes in the examination set;
MFT, a node move frequency table, $MFT[x]$ is the move frequency of node x;
nbiter, the current iteration number;
nbmax, the maxmum number of null iterations;
bestiter, the number of iterations since the latest best solution was found;
optiter, the number of iterations since the latest tabu relaxation;
s, the current solution;
s^*, the minimal solution generated in some iteration from current solution s;
bestsol, the best solution so far;
$NS(s)$, a subset of neighbourhood $N(s)$ or $N^*(s)$ of solution s;
$f(s)$, the value of objective function of solution s. $f(s) = p_0W_0 + p_1W_1 + p_2W_2$, where p_0, p_1, p_2 are the penalties used to weight the classes of conflicts, and W_0, W_1, W_2 are the numbers of first-order, second-order and third-order conflicts:

i.e. simultaneous exams, consecutive exams and exams with only one free period between them;
V^*, a set of nodes having at least one conflict in current solution s;
V, all nodes in graph (i.e. all examinations);
T_i, timeslot i (or period i);
m_i, any feasible atomic move (x, T_i, T_j) that moves a node x from T_i into T_j $(j \neq i)$;
r, a ratio used to limit the uphill movement during TS.

7 The Complete System

We have constructed a multi-phase system to optimize the examination timetabling procedure. This can find better solutions than a single-phase one.

Phase One:

- Analyse the examination set with the quantitative analysis methods described earlier in Section 5 and estimate the number of the lightest exams;
- Apply the bin packing algorithm to generate the initial solution (with a given number of timeslots);
- Evaluate this initial solution. If the initial solution is feasible (i.e. there are no first-order conflicts), go to phase three; otherwise go to phase two.

Phase Two:

- Use the solution generated in phase one as the initial solution;
- Construct an objective function using only first-order conflicts: $f(s) = W_0$;
- Apply OTTABU to the infeasible solution to generate a feasible solution; if OTTABU cannot generate a feasible solution, then add one more period to the timetable and re-run phase two;
- If a feasible solution is generated then proceed to phase three.

Phase Three:

- Use the solution generated in phase one or phase two as the initial solution. Construct an objective function using first-order and second-order conflicts: $f(s) = p_0 \times W_0 + W_1, \quad p_0 \gg 1$;
- Apply OTTABU to the feasible solution to optimize the solution;
- Go to phase four.

Phase Four:

- Use the solution generated in phase three as the initial solution. Construct an objective function using first-order, second-order and third-order conflicts: $f(s) = p_0 \times W_0 + p_1 \times W_1 + W_2, \quad p_0 \gg p_1 \gg 1$;
- Apply OTTABU to the feasible solution to optimize the solution.

8 Results

There are 771 exams to be assigned into 36 timeslots and each timeslot has 2200 seats available. In order to avoid seating large numbers of students at the end of the examination session (where there are no following exterior edges) the seating for these three timeslots is reduced to 1100. Some quantitative data about the examinations are

- The total number of periods, 36,
- The total number of examinations, 771,
- The total number of students, 14 032,
- The average number of exams per period, 21,
- The number of seats for per period, 2200,
- The total number of seats available, 75 900,
- The total number of student-exam enrolments, 46 899,
- The average number of exams per student, 3.34,
- The average number of student-exams per period, 1303,
- The ratio of seat utilization, 61.79%,
- The average number of students per exam, 60.82,
- The total number of edges in the graph, 18 522,
- The average degree per node, 48,
- The density of conflict matrix, 0.06,
- The total weights of edges in the graph, 140 704,
- The average weights of edges per exam, 182.

The number of lightest exams is estimated as $np = 196$ at the 5% accumulation percentage (see Table 2).

In phase one, we apply a bin packing algorithm to get a feasible solution s_0 such that $(W_0/W_1/W_2) = (0/8686/7844)$. This means that no student needs to take two exams at the same time (first-order conflict), 8686 students have to take two consecutive exams (second-order conflicts) and 7844 students have to take exams having third-order conflicts.

We use the TS technique with both short-term tabu list and longer-term tabu list to optimize this timetable. Since a feasible solution was achieved in phase one, we go directly to phase three, apply OTTABU to optimize W_0/W_1 and obtain the result shown in Table 3. We set $p_0 = 500$, $p_1 = 1$, $np = 196$, and $l_{TS} = 9$. l_{TL} was set to 190 in pass 1 and pass 3. The initial solution is characterized by 0/8686/7844 obtained by the bin packing algorithm of phase one ($f(s) = 8686$).

Then, going to phase four, we use the solution s obtained above as an initial value. We set $p_0 = 250\,000$, $p_1 = 500$, $p_2 = 1$, $np = 196$ (at the 5% accumulation percentage), $l_{TS} = 9$, and the initial solution has 0/496/5643, obtained in phase three ($f(s) = 253\,643$).

The best solution is *bestsol* where $(W_0/W_1/W_2) = (0/494/4354)$ which had an execution time of approx. 86 min on a Pentium 100 MHz PC. In this solution, 494 students (approx. 3% of the total enrolment) have to take two consecutive exams. This is the best solution obtained by the system with these data.

Table 3. Case 1: both longer-term and short-term memory applied (phase three)

Pass	Neighbourhood	l_{TL}	$nbmax$	Iterations	Time	$bestsol$	$f(bestsol)$
1	$N^*(s)$	190	570	2595	0:16:00	0/776	776
2	$N(s)$	63	94	761	0:12:06	0/592	592
3	$N^*(s)$	190	570	500*	0:02:14	0/590	590
4	$N(s)$	63	94	1224	0:19:36	0/496	496
Total				5080	0:49:56		

* Because V^* in pass 3 is less than l_{TL}, both l_{TL} and $nbmax$ were changed automatically.

Table 4. Case 2: both longer-term and short-term memory applied (phase four)

Pass	Neighbourhood	l_{TL}	$nbmax$	Iterations	Time	$bestsol$	$f(bestsol)$
1	$N^*(s)$	190	570	772	0:12:15	0/496/4546	252546
2	$N(s)$	63	94	94	0:02:18	0/496/4546	252546
3	$N^*(s)$	190	570	664	0:19:47	0/494/4354	251354
4	$N(s)$	63	94	94	0:02:17	0/494/4354	251354
Total				1624	0:36:37		

9 Comparison

In order to examine the usefulness of longer-term frequency-based memory, we have done some experiments that attempt to explore the behaviour of OTTABU when the longer-term memory is disabled or set with different maximum lengths.

First, the longer-term tabu list in OTTABU was disabled; the penalties p_0, p_1 and p_2 were set to the same values as in cases 1 and 2; $nbmax$ was fixed at 270. We set $p_0 = 500$, $p_1 = 1$, $l_{TS} = 7$ and the initial solution is 0/8686/7844. The results are shown in Table 5.

Table 5. Case 3: short-term memory only. I

Pass	Neighbourhood	$nbmax$	Iteration	Time	$bestsol$	$f(bestsol)$
1	$N^*(s)$	270	785	0:06:00	0/892	892
2	$N(s)$	270	494	0:07:00	0/772	772
3	$N^*(s)$	270	307	0:02:00	0/759	759
4	$N(s)$	270	286	0:04:38	0/748	748
Total			1772	0:19:38		

We have repeated the search, using the same conditions of case 3 with the sole difference that the order of applying neighbourhood definitions is reversed, resulting in a solution initially slower but better than previously obtained. But in the end, the solution quality is about the same, as is the time required to obtain

it. We set $p_0 = 500$, $p_1 = 1$, $l_{TS} = 7$ and the initial solution is 0/8686/7844. The results are shown in Table 6.

Table 6. Case 4: short-term memory only. II

Pass	Neighbourhood	*nbmax*	Iterations	Time	*bestsol*	*f*(*bestsol*)
1	$N^*(s)$	270	767	0:12:16	0/801	801
2	$N(s)$	270	342	0:02:01	0/776	776
3	$N^*(s)$	270	291	0:04:44	0/760	760
4	$N(s)$	270	390	0:02:20	0/749	749
total			1790	0:21:21		

The results show that:

1. By using $N(s)$, we can get a much better solution than by using $N^*(s)$ in pass 1 even though the former took twice as much time as does the latter;
2. The four-pass algorithm can greatly improve the timetable. The best solution obtained in pass 1 is improved in the later passes. The value of the objective function is minimized by about 16% in case 3, and about 7% in case 4;
3. Even though we set a bigger value for *nbmax*, the search ended much earlier than we expected because of cycling. In fact, we have later used a smaller *nbmax* to save time without having a negative impact on the search.

Comparing cases 3 and 4 with cases 1 and 2, we notice that the use of longer-term memory had a greater effect on the quality of the final result. The use of longer-term memory resulted in a solution having a value of 496 whereas without longer-term memory, the best obtainable solution had a value of 748. The reduction of $(748 - 496)/748 = 0.34$ is due entirely to the use of longer-term memory.

We have also set different values for the maximum length of the longer-term tabu list, and find the right l_{TL} not only to get the best solution and but also to reduce search time.

It can be seen from Table 7 that reducing l_{TL} to 30 results in solutions that are much worse than those shown in Table 4, (solution value is 326 668 as opposed to 251 354). We set $p_0 = 250\,000$, $p_1 = 500$, $p_2 = 1$, $l_{TL} = 30$, $l_{TS} = 9$, and the initial solution is 0/8686/7844.

We set $p_0 = 500, p_1 = 1$ and $np = 354$ (at 15% accumulation percentage), $l_{TS} = 9$, and the initial solution is 0/8686/7844 obtained by a bin packing algorithm ($f(s) = 8686$). The result is shown in Table 8.

Thus far we have examined the influence of the longer-term tabu list on the Ottawa University data used. A similar study was performed on some data available in Carter's repository [11]. These data sets, called CAR-F-92 and UTA-S-92 in Table 1 of [11], were used as the input data of OTTABU. In both cases, when the longer-term list was removed from the program the results were not

Table 7. A longer-term tabu list that is too small

Pass	Neighbourhood	l_{TL}	$nbmax$	Iterations	Time	$bestsol$	$f(bestsol)$
1	$N^*(s)$	30	270	2513	0:39:47	0/812/4745	470745
2	$N(s)$	30	270	768	0:22:06	0/659/4644	334144
3	$N^*(s)$	30	270	579	0:08:40	0/651/4817	330317
4	$N(s)$	30	270	839	0:19:33	0/644/4688	326668
Total				4699	1:29:05		

Table 8. A longer-term tabu list that is too large

$Pass$	$Neighbourhood$	l_{TL}	$nbmax$	$Iterations$	$Time$	$bestsol$	$f(bestsol)$
1	$N^*(s)$	354	1062	2595*	0:16:32	0/776	776
2	$N(s)$	118	177	844	0:14:12	0/592	592
3	$N^*(s)$	354	1062	502*	0:02:29	0/590	590
4	$N(s)$	118	177	1425	0:23:49	0/509	509
Total				5080	0:55:52		

* Both l_{TL} and $nbmax$ are changed automatically when the size of V^* is less than np.

as satisfactory as when it was present. For the CAR-F-92 data set, the use of a longer-term list reduces the ultimate cost by about 15% and for the UTA-F-92 data set, the figure is about 7%. Both these values are less than the value calculated in the previous section for the Ottawa data.

We believe that the main reason for this is related to the matrix density. Both the CAR and UTA data sets have a matrix density that is relatively high, 0.14 and 0.13 respectively. The density of the Ottawa data was calculated to be 0.06. Thus there are less than half as many edges per node on average for the Ottawa data as there are for the others.

Thus, when a move is made using the Ottawa data a large reduction in the penalty function is possible because the matrix density is smaller, increasing the likelihood that a move of a node to the target timeslot will result in a smaller (or zero) penalty. Study of this effect is continuing.

10 Comparison of OTTABU with Other Results

The comparison of various strategies within an environment in which everything else is held constant is relatively easy. This is the approach followed in the preceding sections in which it became evident that incorporating constraint relaxation and a longer-term tabu list into a TS led to better-quality solutions than could be obtained otherwise, at least in the environment used. It is something else to decide whether these techniques will also work in other environments.

This question is complicated by the fact that many of the other algorithms have been written to solve certain problems particular to some institution. As Carter et al. [11] point out, “As all real timetabling problems have different

side constraints with them, for all practical purposes each school has a unique problem."

Nonetheless, in order to see whether the improvements demonstrated by OTTABU in the Ottawa University environment would continue to be shown in other environments, some changes were made to the algorithm and the data in such a way that some comparisons could be made. These included the following modifications:

1. Data from two other universities were downloaded from Carter's repository [11]. The data sets, CAR-F-92 and UTA-S-92, were used as the input data of OTTABU.
2. The value of the objective function was changed to include farther proximities or higher-order conflicts. The cost weights were taken as $w_1 = 16$, $w_2 = 8$, $w_3 = 4$, $w_4 = 2$, $w_5 = 1$. The possibility of simultaneous examinations was taken as a hard constraint and was forbidden.
3. The common metric "cost", taken to be the total penalty using the weights listed above divided by the number of students involved, was used as the basis of comparison.
4. OTTABU was modified to incorporate a random tenure in its short-term tabu list so that it could be restarted with the same parameters several times to get a number of final solutions and a selection of best solutions and costs. This also makes the short-term tabu list part of OTTABU more like the algorithm of Di Gaspero and Schaerf [12].

With these modifications several runs were made and the final solutions obtained by OTTABU could be compared directly with the results published by Carter et al. [11] and by Di Gaspero and Schaerf [12]. The program EXAMINE, reported by Carter et al. [11] uses 40 different strategies overall, incorporating five basic sorting rules, using cliques or not, using costs or not and using backtracking or not. The results published in their Table 5 have associated costs ranging from 6.2 to 8.2 for the data set CAR-F-92 and ranging from 3.5 to 6.4 for data set UTA-S-92.

The program of Di Gaspero and Schaerf uses a short-term memory with tenure varying randomly between 15 and 25. The stopping criteria is based on the number of iterations from the last improvement (idle iterations), the number of iterations depending of the instance, varying from 2000 to 20 000. Their average cost for CAR-F-92 was 5.6 and for UTA-S-92 was 4.5.

In comparison, the results obtained by OTTABU as modified above were an average of 4.7 for CAR-F-92 and 4.0 for UTA-S-92. These results are shown in Table 9.

The average cost of the OTTABU solution is less than that reported by Di Gaspero and Schaerf in both cases and is less than all but one of the values reported by Carter et al. for these two data sets.

The overall conclusion resulting from this comparison is that OTTABU's use of longer-term tabu lists has a demonstrated beneficial effect in all the cases tested and that OTTABU's results compare favourably with the other algorithms examined.

Table 9. Comparison of results

Data set	Number of timeslots	OTTABU average	Di Gaspero and Schaerf average	Carter et al.
CAR-F-92	32	4.7	5.6	6.2–8.2
UTA-S-92	35	4.0	4.5	3.5–6.4

11 Conclusion

In the OTTABU algorithm, we use both a recency-based short-term tabu list and a move (or transition)-frequency-based longer-term tabu list which prevent cycling and diversify the search space effectively to help get a better solution. We have developed a four-pass TS technique to intensively search the region containing the best solution in order to search for a better solution. Based on our experimental tests using real data from three institutions we conclude:

1. For a large-scale examination timetabling problem, TS techniques, based on short-term memory only, make progress quickly from a non-optimal initial solution but cannot generate a really high-quality solution. We have implemented an algorithm in which both longer-term and short-term memory are used to generate better solutions than a short-term memory alone. This is observed to be true on every data set we have used.
2. It is shown in our study that the move- or transition-frequency-based long-term memory is a technique that is easy to understand and implement. The TL tabu list was observed to forbid the movement of potential overactive exams, thus preventing cycling and diversifying the search space.
3. We successfully applied a quantitative analysis procedure to estimate the number of the lightest exams (i.e. the potentially movable exams). This makes it possible to automatically determine the appropriate size of the longer-term tabu list.
4. The relaxation of tabu lists is an effective way of avoiding potential over-uphill movements, reduce the search time and search space and increase the likelihood of finding optimal solutions.
5. The four-pass algorithm OTTABU is an effective mechanism to control the balance between diversification and intensification in the solution search.

Acknowledgements. The former registrar of the University of Ottawa, Mr George H. von Schoenberg, was instrumental in initiating the research that led to this study. The Manager, Registration and Records, Ms Pauline Bélanger, helped greatly in all facets of the implementation and data collection.

References

1. Peck, J.E.L., Williams, M.R.: Algorithm 286 – Examination Scheduling. Commun. A.C.M. **9** (1966) 433–434

2. White, G.M. Chan, P.-W.: Towards the Construction of Optimal Examination Schedules. INFOR **17** (1979) 219–229
3. Carter, M.W., Laporte, G., Chinneck, J.W.: A General Examination Scheduling System. Interfaces **24** (1994) 109–120
4. Carter, M.W., Laporte, G.: Recent Developements in Practical Examination Timetabling. Lecture Notes in Computer Science, Vol. 1153. Springer-Verlag, Berlin Heidelberg New York (1996) 3–21
5. de Werra, D., Hertz, A.: Tabu Search Techniques – a Tutorial and Applications to Neural Networks. OR Spektrum **11** (1989) 131–141
6. Hertz, A.: Tabu Search for Large Scale Timetabling Problems. Eur. J. Oper. Res. **54** (1991) 39–47
7. Boufflet, J.P., Negre, S.: Three Methods Used to Solve an Examination Timetable Problem. Lecture Notes in Computer Science, Vol. 1153. Springer-Verlag, Berlin Heidelberg New York (1996) 3–21
8. Ferland, J.A.: Generalized Assignment-Type Problems: a Powerful Model Scheme. Lecture Notes in Computer Science, Vol. 1408. Springer-Verlag, Berlin Heidelberg New York (1998) 53–77
9. Glover, F: Tabu Search: a Tutorial. Interfaces **20** (1990) 74–94
10. Glover, F., Laguna, M.: Tabu Search. Kluwer, Dordrecht (1997)
11. Carter, M.W., Laporte, G., Lee, S.Y.: Examination Timetabling: Algorithmic Strategies and Applications. J. Oper. Res. Soc. **47** (1996) 373–383
12. Di Gaspero, L., Schaerf, A.: Proc. 3rd Int. Conf. Practice and Theory of Automated Timetabling (Konstanz, Germany) (2000) 176–179