INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Université d'Ottawa · University of Ottawa

Investigations of the Longer Term Memory, Relaxation and Restructure in the Tabu Search Heuristic Optimization of Examination Timetables

Supervised by Dr. George White

Siqun Xie
Student Number: 1887435
Candidate for Master Degree in Computer Science

School of Information Technology and Engineering
University of Ottawa

Jun 10, 2002

To my wife Hong Yang, and
my sons, Yangda and Yangde

Abstract

The examination scheduling problem is a difficult problem with NP-hard complexity and several methods for heuristically solving it have been developed over the past decades. This thesis investigates the casting of examination timetables with tabu search techniques, the combination of the move or transition recency based short-term memory and move frequency based longer-term memory and several tabu search variations such as solution restructuring, tabu relaxation, and alternating neighborhoods. By integrating the tabu techniques mentioned above, a multi-phase system OTTASYS has been implemented and tested on several sets of real data. It is shown here that the quality of the solutions obtained compare favourably with other published algorithms when presented with the same data. As the engine of the system, the algorithm OTTABU is proven powerful and productive and also can be used as a platform for further research.

Acknowledgement

I would like to express my appreciation to Dr. George White, without his supervision, encouragement, and patience, I could not finish this research and thesis writing.

I would like to thank many people who provided help during the preparation and completion of this thesis.

Contents

Introduction

The Examination Timetabling problem is a difficult combinatorial exercise and all published Tabu Search applications on examination timetabling are based on a list of recent moves or solutions (called recency-based tabu list or a short-term memory based tabu list). Even though these applications can generate good quality solutions, the search process used is more prone to develop cycling and cannot diversify the search space efficiently because of the recency-based short-term memory. As Glover suggested [Glover90], "the integration of both recency- and frequency-derived factors in human long-term memory suggests that determining an effective combination of the two may provide better than either in isolation." Inspired by this, we implemented a multi-phase system OTTASYS to improve solution quality.

The OTTASYS contains a solution restructuring mechanism and a core subsystem or algorithm named OTTABU. In the OTTABU algorithm, the recency-based short-term memory and transition frequency-based longer-term memory Tabu Search techniques are combined. More criteria are integrated in the algorithm for selecting the local solutions. It also provides a controlled tabu relaxation mechanism and neighborhood alternating technique. The OTTASYS has been tested on real examination data and has produced solutions which are better than those previously published.

Many studies have been done on the Examination Timetabling problem for many years, dating back to the early 1960's [PeckW66]. The study of this problem has led to several implementations that have been used very successfully at many universities. Much of the early work was based on bin packing algorithms with some heuristic rearrangement of the schedule. This was based on some version of the

travelling salesman's algorithms which reduces 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 also considered a Graph Coloring problem. In recent years, some researchers have applied Tabu Search techniques to solve Examination Timetabling problems. A review of the majority of the work published in this area can be found in the comprehensive coverage of Carter and Laporte [CarterL96]. The key points of this work are presented in Chapter 1.

Tabu Search is considered to be part of the class of algorithms called Generalized Search [CarterL96] or Neighborhood Search [Newell99]. This includes Simulated Annealing, Decent Search, and Genetic Method. It begins with one or more initial solutions, then the neighborhood of that solution is calculated and moved to one of the neighbors. The neighbors of a solution can be generated in different ways. We can use an example to explain this concept. Assume we have an examination timetable denoted by s_0. By employing some rules, we choose an exam from some period (or timeslot), and move it into another period to generate a new timetable s_{01}. The new timetable s_{01} is referred to as a neighbor of s_0. The set or subsets of all possible neighbors of s_0 are referred to as the neighborhood of s_0. These neighborhood search algorithms employ different strategies to generate neighborhoods that try to avoid getting stuck in local solution areas and investigate a more global region of the space. Generally speaking, this search process cannot guarantee to generate the best solution but it does generate better solutions than other classes of algorithms in practice. Because of this characteristic, we treat them as heuristic methods. The Tabu Search memory structures, strategies, and principles are reviewed in Chapter 2.

The first application of Tabu Search techniques on the examination timetabling problem occurred about 15 years ago. Hertz and de Werra applied a Tabu Search technique to the graph colouring problem [HertzW87], and later to both the examination and the course timetabling problem in 1991[Hertz91]. Their colleague,

Costa, also did similar research on the course timetabling problem [Costa94]. In 1996, Boufflet *et al* completed an application of Tabu Search technique for the examination timetabling problem at the University of Technology of Compiègne in France [BouffetS96]. Other examples of this research are Di Gaspero and Schaerf 's work, where the advantages of Tabu Search techniques are shown in comparison to other algorithms for examination timetabling problems [GasperoS00]. All this research is based on recency-based short-term memory Tabu Search techniques. Their work is detailed in Chapter 3.

Chapter 4 discusses a commonly used graphical representation of the examination set by which the conflict relations between examinations are very simply reflected. Then the graph vertex coloring concept is applied to the scheduling of the examination timetable. Several considerations of typical issues such as the types of atomic moves and simplification of computation in the local search, weighting of conflicts between timeslots, the implementation of pre-assigned examinations, and timeslot distancing, are discussed as well. With this modeling of examination timetables, we can start the Tabu searching.

Details of the OTTABU algorithm are presented in Chapter 5. A quantitative analysis of the examination set to be scheduled provides an estimation of the size of the tabu list *TL*, which is constructed and adjusted dynamically by calculating transition frequency of each exam during life-time searching. The quantitative method is developed in Chapter 6.

As a subsystem of the multi-phased OTTASYS, the OTTABU is executed multiple times in the complete system, with different penalty sets in each phase. Solution restructuring, as an application of strategic oscillation technique, can drive the search into a promising space. We have done many experiments to prove that we can benefit from this system. The chapters 7, 8 and 9 give the details of the experiments and results, which demonstrate that this multi-phase system can generate much better solutions than a single one phase system. In particular, we have tested each

individual main component or mechanism of the OTTASYS and the OTTABU to investigate their impacts. Some of these components or mechanisms are: solution restructuring, two penalty sets (second order conflicts only versus up to six order conflicts); two types of neighborhood alternation; tabu relaxation activation versus deactivation; and so on.

There are still a lot of interesting search spaces for discovery on this topic for the future. I would like to continue my research on following topics: the rejected elite solution chain and the impact and behavior of heavier exams in Tabu Search process.

Chapter 1

The Examination Timetabling Problem

The examination timetabling problem is a difficult combinatorial problem that is faced by educational institutions and also is an interesting intellectual exercise that has been studied for many years dating back to at least the early 1960s [PeckW66]. This work has led to several implementations that have been used with much success at many universities [WhiteC79] [CarterLC94]. Carter and Laporte have given comprehensive reviews of most of the work published in this area in the recent 30 years in two papers [Carter86] [CarterL96].

In this chapter, some topics are presented such as what an examination timetable problem is, which main research methods are used to pursue the solutions, and what issues the researchers are facing.

§1.1 Primary Objectives and the First and Second Order Conflicts

The basic examination timetabling problem is very easy to describe but not easy to define. For a feasible examination timetable, obviously no student can write more than one examination at the same time. Beside this, we may have to consider to meet other constraints, *e.g.* the number of periods available and the maximum seats available per period. A constraint is a hard one if it must be met, otherwise it is a soft one. In order words, an examination table is still feasible if soft constraints are violated.

We define a hard constraint to be the *first order conflict* if it is evaluated in the objective function. Any occurrence of a student writing more than one examination at the same time is a first order conflict. For such a first order conflict, a penalty is

imposed. For other constraints like the maximum number of exams and maximum seats available per timeslot, we do not integrate them into the objective function, but treat them as hard constraints which must not be violated. We call an examination timetable infeasible if it either has first order conflicts or violates any hard constrains.

We define a soft constraint to be *a second order conflict* if it is considered and evaluated in the objective function. In our investigation, we focus on these examinations which conflict and are scheduled in consecutive periods or are scheduled into periods separated by less than six periods.

The primary objective of solving the examination timetabling problems is to satisfy hard constraints while minimizing the second order conflicts.

For some institutions, students are given a wide degree of latitude in their course selections and it is becoming more and more difficult to control the length of the examination timetable. The primary objective not only is to find a feasible examination timetable but also to reduce the number of required time periods [CarterL96]. In contrast, in some other institutions, it is not difficult to find a feasible examination timetable for a given period, and the primary objective may be to maximize students' study time in examination season [Bulln98].

In some cases, this first order conflict constraint needs to be relaxed, and the objective is to minimize the second order conflicts. For example, this situation has been discussed here at University of Ottawa. If there are a lot of students who have second order conflicts because only one (or two) students would have a first conflict in a timetable then the assignment that gives the one or two students the job of writing two exams at once is preferred if the other students don't have to write consecutive exams. What happens in these cases is that the one or two students write the exam at a special time (different from that of the other students). The exam may be changed also. This situation also has been reported by Burke [BurkeEF96].

§1.2 Practical Constraints

In reality, we have to manipulate many side constraints while generating a feasible examination timetable for schools, colleges, and universities. Usually, the following side constraints need to be considered in practice [CarterL96]:

1. Certain groups of exams may be required to take place at the same time;
2. Precedence constraints between exams restrict the ordering;
3. Certain exams are required to be in consecutive periods;
4. Certain exams are required to be on the same day;
5. Rules of the form "No student can write x exams in any y consecutive periods";
6. Certain periods are excluded for an examination (e.g., an exam may be restricted to "evening periods only");
7. Certain (groups of) students are restricted to a subset of the available periods (note that this is a simple extension of excluding exams from certain periods);
8. There is a limit on the total number of students who write examinations at the same time (total seats);
9. There is a limit on the number of examinations that can occur at the same time (e.g., number of rooms or invigilators).

There may also be a variety of constraints with respect to resources and examiners associated with each examination:

10. Limited number or size of rooms available, e.g., multiple exams in one location, exams may be split over several geographically similar locations, exams may have specific room or building preferences, exams may be pre-assigned or excluded from certain rooms;
11. Special resource requirements (large desks, TV, computers, etc.);

12. Schedule examiners (no examiners should have two exams on the same day; or examiners with two exams should have them in nearby locations).

13. Assign invigilators to exam rooms (minimize number of invigilators depending on the actual number of students in the room);

14. For special students, need to specify exam spacing, length, location, and special resources.

However, there are a variety of softer secondary objectives some schedulers may need to consider, but if the cost is too much, they could be discarded:

15. Minimize the number of occurrences of x examinations in any y consecutive periods for all students;

16. Spread examinations out as evenly as possible (for each student);

17. Try to schedule all examinations as early as possible. (Note that this criterion is directly contrary to some of the objectives above);

18. Retain the optimized timetable from the previous year as much as possible. This last criterion is not very appropriate unless there is very little change in student demand patterns from one year to the next.

19. Maximize flexibility: maximize the number of exams that can be moved to another time period without disrupting the rest of schedule.

There are more. However, this shows the wide variety of different factors that any general system would have to deal with. Burke *et al.* conducted a survey of timetabling officers and administrators from institutions throughout the UK on the examination timetabling problem [BurkeEF96]. One of the findings that was reported was that the various institutions surveyed placed differing amounts of importance on the same constraints. This is inevitable given differing circumstances. For instance an institution with a longer timetable may give more weight to student

load considerations, while an institution with a very condensed timetable would find these much harder to satisfy. A rating table is shown in Table 1.1.

No	Constraints	Importance
1	There must not be more students scheduled into a room than there are seats	*94%*
2	Exams with questions in common must be scheduled in the same period	*87%*
3	Some exams may only be scheduled within a particular set of periods	*69%*
4	Only exams of the same length may be scheduled in the same room	*63%*
5	Exams with the most students must be scheduled early in the timetable	*60%*
6	Some exams must only take place in particular rooms	*58%*
7	Large exam halls must be scheduled in preference to small ones	*54%*
8	Exam A must be scheduled before exam B	*52%*
9	No student is scheduled to exams in two consecutive periods	*48%*
10	No student is scheduled to write more than one exam in any particular day	*38%*
11	Each student's exams must be evenly spread throughout the timetable	*17%*
12	No student is scheduled to exams in two consecutive days	*17%*
13	Exams must be scheduled into rooms near to the relevant department	*10%*

Table 1.1 Examination Scheduling Constraints and Their Relative Importance

Hence, some constraints must be treated as hard constraints, while some constraints must be weighted appropriately and integrated into the objective functions or evaluation forms in order to optimize timetables toward good quality.

But, for lack of common models dealing with constraints, it is very difficult to evaluate and compare examinations timetables if not possible.

§1.3 Algorithms and Approaches

Carter et al [CarterL96] examined the researches and approaches on examination scheduling in the past decade, divided the approaches into four types, and listed concrete algorithms in each type as shown in Table 1.2. They found that most algorithms only address a subset of the constraints and objectives of the general examination scheduling problem. The reason may be either some difficulties in the implementation or a trade-off between speed and comprehensiveness, for instance, in GA or Tabu Search algorithm, complex evaluation functions may greatly increase the speed required to evaluate lengthy expressions and comparisons.

Cluster methods	
Description	The set of exams are first combined into blocks or groups corresponding to a set of compatible examinations, *e.g.*, this group can be feasibly be assigned to some period with no (or only few) conflicts. In a second phase, the clusters are sequenced into specific periods to minimize some objective or satisfy some constraints.
Algorithms	*Largest ones first, Smallest ones first, and Hybrid*
Sequential methods	
Description	Exams are assigned to a specific period one at a time, chosen using some sequencing strategy. There exist a wide variety of approaches distinguished by the order for selecting the next exam, and the way the period is chosen. A two-phase strategy is commonly used. A construction phase produces an initial solution, and an improvement phase makes modifications and improvement.
Algorithms	*Largest degree, Saturation degree, Largest weighted degree, Largest enrollment, Largest number of papers, User defined priority groups, and Random ordering*
Generalized Search Strategies (Neighborhood Search)	
Description	They begin with one or more initial solutions, and employ a search strategy that tries to avoid getting stuck in local solution areas, and investigates a more global region of the space. These techniques all share the property that, if they are run long enough, they might discover optimal solutions.
Algorithms	*Simulated Annealing, Tabu Search, Genetic Algorithm, and Decent Search*
Constraint based Approaches	
Description	These approaches model a problem as a set of variables with a finite domain, to which values must be assigned so as to satisfy a number of constraints.
Algorithms	*Constraint satisfaction problems and Constraint logic programming*

Table 1.2 Approaches Addressing the Examination Scheduling Problems

Chapter 2

Tabu Search : Concepts, Principles and Applications

The basic form of Tabu Search techniques was created in the late 1960s by Fred Glover, and since then, he and many other researchers have developed and applied Tabu Search techniques for optimization in many fields such as planning and scheduling, telecommunication routing, transportation, parallel computing, neural networks and learning, graph, manufacturing, etc. The philosophy of Tabu Search is to derive and exploit a collection of principles of intelligent problem solving and based on selected concepts that unite the fields of artificial intelligence and optimization.

§2.1 Tabu Search Concepts

There is no formal definition to describe this method. In fact, unlike some algorithms, Tabu Search techniques are kinds of heuristic approaches. There are no strict or universal rules and algorithms to guarantee the generation of high quality optimal solutions, even though Tabu Search applications can find better solutions in the fields mentioned above. Tabu Search can be described as a set of mechanisms, approaches, guidelines, principles and strategies to prevent optimization processes from becoming trapped at locally optimal solutions and guide processes to cross boundaries of feasibility or local optimality back and forth, and terminate processes when a preset goal is reached. One of these components is its use of adaptive memory, which creates a more flexible search behavior to guide the moves in the local search. Actually, the memory-based strategies are the hallmark of Tabu Search approaches. Because Tabu Search provides such high level strategies to guide and modify other heuristics to produce solutions beyond local optimality, Glover calls it

a meta-heuristic approach. Let us review some fundamental concepts, strategies and guidelines used in Tabu Search.

Model and Objective Function

Given a problem which can be modeled, we can build an objective function (or a set of objective functions) to evaluate the quality of a solution of the problem. The process of problem solving is equivalent to the optimization (minimizing or maximizing) of the objective function. Without losing generality, we apply Tabu Search to minimize an objective function $f(x)$, subject to $x \in X$, where $f(x)$ may be linear or nonlinear. The set X contains all solutions that satisfy predefined constraints of a problem.

Move and Transition Neighborhood

Tabu Search begins in the same way as ordinary local or neighborhood search, proceeding iteratively from one point (solution) to another until a chosen termination criterion is satisfied. Each $x \in X$ has an associated neighborhood $N(x) \subset X$, and each solution $x' \in N(x)$ is reached from x by an operation called a move. The move is a transition from a solution to a transformed solution. A starting point for Tabu Search is to note that such a move may be described by a set of one or more attributes (or elements), and these attributes (properly chosen) can become the foundation for creating an attribute based memory.

Local Optimum and Descent Method

In order to minimize an objective function $f(x)$, a simple descent method only permits moves to neighbor solutions that improve the current objective function value and ends when no improving solutions can be found. A pseudo-code of a generic descent method is presented in Figure 2.1. The final x obtained by a descent method is called a local optimum, since it is at least as good as or better than all solutions in the neighborhood. The evident shortcoming of a descent method is that

such a local optimum in many cases will not be a global optimum, i.e., it usually will not minimize $f(x)$ over $x \in X$.

The version of a descent method called steepest descent scans the entire neighborhood of x in search of a neighbor solution x' that gives a smallest $f(x')$ value over $x' \in N(x)$. Steepest descent implementations of some types of solution approaches (such as certain path augmentation algorithms in networks) are guaranteed to yield globally optimal solutions for the problems they are designed to handle, while other forms of descent may terminate with local optima that are not

---

1) Choose $x \in X$ to start the process.

2) Find $x' \in N(x)$ such that $\min\{f(x') < f(x)\}$.

3) If no such x' can be found, x is the local optimum and the method stops.

4) Otherwise, designate x' to be the new x and goto 2.

Figure 2.1 Generic Descent Method

global optima. In spite of this attractive feature, in certain settings steepest descent is impractical because it is computationally too expensive, such as when $N(x)$ contains many elements or each element is costly to retrieve or evaluate. Still, it is often valuable to choose an x' at each iteration that yields a "good" if not smallest $f(x')$ value.

Uphill and Short Term Memory

In order to avoid getting stuck in a local optimum, we accept an uphill move, which means we may pick a new solution worse than the current one. If we accept that move, obviously, we need a mechanism to guide search that the next move can not

step back down to the position vacated. In other words, we need to memorize the recent camp-sites we have visited. This is the recency-based short-term memory. A Tabu Search algorithm is listed in Table 2.2. Note that, in this algorithm, a short-

1. *Let $x_0 \in X$ be an initial solution, $x^* \in X$ denote the best solution generated so far, Lx is a tabu list to store the recent solutions;*
2. *Initialize $Lx = \Phi$; $x^* = x = x_0$;*
3. *Choose $x \in X$ to start the process.*
4. *Find all $z \in N(x)$-Lx and $z \in Lx$ such that $f(z) < f(x^*)$.*
5. *Determine x' such that $f(x') = \min \{f(z)\}$,*
6. *If $f(x') < f(x^*)$, then $x^* = x'$.*
7. *$x = x'$; add x' into Lx and drop the oldest one if the Lx is full.*
8. *if $f(x) \leq$ LOWBOUND or after a maximal number of iterations, stop*
9. *else goto 4 /* continue next iteration */*

Figure 2.2 A Generic Tabu Search Method

term tabu list *TS* or *Lx* is used as a safeguard against cycling, and a tabu add-and-drop rule should be defined.

Memory Structure

The local search strategies are often memoryless ones that keep no record of their past moves or solutions. The Tabu Search meta-heuristic makes abundant use of memory in two ways: the short-term memory and longer-term memory. 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 [GloverL97] emphasize four important dimensions of the memory structures used in Tabu Search. These are:

Recency - Tabu Search 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 - Tabu Search calculates ratios that keep track of which attributes move the most and how often they move.

Quality - In principle, Tabu Search can distinguish a better solution from a worse one by using criterion other than a single objective function, *i.e.*, Tabu Search can directly incorporate multiple-criteria decision capability.

Influence - Tabu Search uses the information it has in its memory to evaluate the choices it will make and the quality of the solutions it finds. The notion of influence does not simply refer to anything that creates a "large change", however, but rather integrates the two key aspects of diversification and intensification in Tabu Search by seeking change that holds indication of promise.

Intensification and Diversification

The strategies used by Tabu Search can be classified as using intensification and/or diversification. 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 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 behavior 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 mined.

§2.2 Guidelines

Glover gave seven guidelines [Glover90] in order for applications to benefit from the Tabu Search heuristic methodology:

1. When tabu restrictions are based on a single type of move attribute, it is generally preferable to select an attribute whose tabu status less rigidly restricts the choice of available moves. This guideline deserves to be qualified under certain circumstances. It appears preferable to avoid (or supplement) a tabu restriction whose degree of flexibility encourages the generation of consecutive solutions that are all accessible to a common earlier solution.

2. Incorporate separate but parallel lists for different attribute types, where the sizes of these lists reflect the relative differences in constraining the number of available moves.

3. Embody the treatment of aspiration criteria in an attribute-based framework analogous to that used to define tabu restrictions (employing the same or different attribute types).

4. To combine the diversification and intensification goals, create a rating system by refering to target analysis, maintaining records of highly rated attributes which appear in solutions generated. On iterations where conditions derived from target analysis disclose standard evaluations to be reliable, supplement these evaluations by favouring attributes with high historical ratings that less frequently occurred in previous solutions.

5. Devote a limited number of preliminary iterations to identify historical statistics for move evaluations in each distance class, and apply relationships from target analysis to obtain effective choices at subsequent iterations (by comparing the values of associated single iteration parameters to these statistics).

6. For longer-term diversification, employ frequency-derived penalties to drive the search away from solutions previously encountered either by restarting or by progressing from the current solution.

7. The integration of both recency- and frequency-derived factors in human long-term memory suggests that determining an effective combination of the two may provide better solutions than either in isolation.

§2.3 Principles

During the development of Tabu Search techniques in the past decade, these guidelines listed in the previous section have evolved into some principles. Glover *et al* [GloverL97] presented the following principles which are important for an effective application of the Tabu Search methodology and they merit fuller investigation. According to empirical studies, the considerations in the implementation of Tabu Search change from application to application.

The principle of persistent attractiveness

This principle can be expressed as follows. If there are some attributes which are not considered or not weighted enough in an evaluation mechanism in a Tabu Search algorithm, an inducement to evaluation should be given to them to upgrade the chance they will be selected if these attributes persist in being attractive, as a result of belonging to moves with high evaluations. For same reason, these attributes can be dropped from or their weight can be reduced in an evaluation mechanism if the persistent attractiveness does not last. On the other hand, regarding to the fact that attributes which persisted in being attractive may not necessarily be attractive in

combination, care must be taken in the application of this principle to avoid bringing in other such attributes that may cause a contrasting change in structure or a notable deterioration in solution quality.

The principle of persistent voting

While persistent attractiveness refers to evaluations over a span of iterations, persistent voting refers to evaluations within a single iteration, but performed by multiple decision rules. One way to achieve this integration is by establishing a parametric weighing of functions that underlie different evaluations and searching over different parameter values to identify effective choices.

Interlinking: compound move, variable depth and ejection chains

This principle originated from such idea that the issues of influence, and their relevance for combining the goals of intensification and diversification, do not simply concern isolated choices of moves with particular features, but rather involve coordinated choices of moves with interlinking properties. The theme of making such coordinated moves leads to consideration of compound moves, fabricated from a series of simpler components.

An ejection chain is an embedded neighborhood construction that compounds the neighborhoods of simple moves to create more complex and powerful moves. In rough overview, an ejection chain is initiated by selecting a set of elements to undergo a change of state, which leads to identifying a collection of other sets, with the property that the elements of at least one must be ejected from their current states. States change steps and ejection steps typically alternate, and options for each depend on the cumulative effect of previous steps (usually, but not necessarily, being influenced by the immediately previous step). In some cases, a cascading sequence of operations may be triggered representing a domino effect. The ejection chain may provide a unifying thread that links a collection of useful procedures for exploiting

structure, without establishing a narrow membership that excludes other forms of classifications.

The proximate optimality principle

This principle stipulates that good solutions at one level are likely to be found at an adjacent level, and it applies to both simple and compound moves. This proximate optimality notion can be exploited in Tabu Search by remaining at each successive level for a chosen number of iterations, and then restoring the best solution found to initiate a move to the next level. For example, a level may be characterized by compelling values of a function to fall in a given range, or by requiring a given number of elements to be included in a partial construction. Then the process can be controlled to remain at a specified level by employing a neighborhood whose moves maintain the functional values (or number of elements) within the specified limits.

The principle of congenial structures

The key idea is that there often exist particular types of solution structures that provide greater accessibility to solutions of highest quality, and, as an accompaniment, there also frequently exist special evaluation functions (or auxiliary objective functions) that can guide a search process to produce solutions with these structures.

The pyramid principle

This principle rests on the following observation. Consider an arbitrary improving path from a given starting solution to a local optimum. As search gets closer to the local optimum, the tributaries that provide improving paths to other local optima become fewer, since all such paths at any given level are contained among those at each level farther from the local optimum. The pyramid principle can be expressed by saying that the total number of improving paths decreases as the objective function $f(x)$ moves closer to a global optimum.

The space/time principle

The space/time principle is based on the observation that the manner in which space is searched should affect that measure of time. Some searching spaces could consume much more time for Tabu Searching than others, but do not promise to get optimal solutions in them. Hence, it is worth building a monitoring or rating system in a Tabu Search algorithm to detect such things and to avoid to get into them. This principle depends on the connectivity of neighborhood space, and more precisely on the connectivity of regions that successively become the focus of the search effort.

Chapter 3

A Review of Tabu Search Techniques in Solving Examination Timetabling Problems

There are not a lot of papers published about Tabu Search techniques being used to solve the examination timetabling problem. However, we should not conclude that there is reluctance to apply Tabu Search techniques to the problem. In fact, some researchers have found that it is good enough to use recency-based Tabu Search techniques to solve this sort of examination scheduling problem.

Hertz and de Werra applied a Tabu Search technique to the graph colouring problem [HertzW87] and later to both examination and course timetabling problems [Hertz91]. Their colleague, Costa did very similar research on the course timetabling problem too [Costa94]. Boufflet *et al* in 1996 have completed an application of Tabu Search technique to the examination timetabling problem at the University of Technology of Compiègne in France [BouffetS96]. In Di Gaspero and Schaerf's work, the advantages of Tabu Search techniques are shown in the comparison with other algorithms for examination timetabling problems [GasperoS00].

§3.1 Hertz and de Werra

In their paper "*Using Tabu Search Techniques for Graph Colouring*" [HertzW87], the Tabu Search techniques and the application to graph node coloring are described very well. Their work aimed at finding a coloring of a given graph G with k colors.

The problem is expressed as follows: given a graph $G=(V, E)$, a feasible solution s will be a partition of the node set V into a fixed number k of subsets V_1, V_2,..., and V_k, denoted as $s=(V_1, V_2, ..., V_k)$. Let $E(V_i)$ be the collection of edges of G with both endpoints in V_i. An objective function f is used to compute the penalty corresponding

to the number of edges for which both endpoints are in the same V_i (*i.e.*, have the same color) for a solution s:

$$f(s) = \Sigma|E(V_i)|, \quad i=1, \ldots, k$$

Clearly s will be a coloring of the nodes of G with k colors if and only if $f(s) = 0$.

Given a solution (or partition) s, a "neighbor" solution s' (*i.e.*, another partition into k subsets of nodes) can be generated as follows by an atomic move: choose a random node x among all those vertex subsets which are incident to an edge in $E(V_1) \cup E(V_2) \cup \ldots \cup E(V_k)$. Then assuming $x \in V_i$, choose a random color $j \neq i$ and obtain s' from $s=(V_1, V_2, \ldots, V_k)$ by setting:

$$V'_j = V_j \cup \{x\}; \quad V'_i = V_i \setminus \{x\}; \quad V'_r = V_r \text{ for } r=1,\ldots,k; \; r \neq i, j.$$

We have $s' = (V_1, V_2, \ldots, V'_i, \ldots, V'_j, \ldots, V_k)$.

After a number of neighbors of s (which do not lead to tabu moves) is generated, the best one s^* is selected to replace s, *i.e.*, the solution moves from s to s^*. The new search continues from s^* and so on. Now the iterations continue until either a solution s such that $f(s)=0$ is found or the maximum number *nbmax* of allowed iterations is reached. The latter means a coloring has not been obtained.

The tabu list is obtained as follows: whenever a node x is moved from V_i to V_j to get the new solution, the pair (x, i) becomes tabu: node x cannot be returned to V_i for some number of iterations called its tenure. The tabu tenure is set to 7 in their algorithm.

Input:

G=(V,E); k = number of colors; |T|= maximum size of tabu list; nbmax = maximum null iterations; rep = number of neighbors in sample.

Initilization:

Generate a random solution $s=(V_1,V_2,...,V_k)$; choose an arbitrary tabu list T; nbiter:=0;

While** f(s)>0 **and** nbiter<nbmax **do

Generate rep neighbors s_i of s with move $s \rightarrow s_i \notin T$ or $f(s_i) \leq A(f(s))$ (as soon as an s_i with $f(s_i) < f(s)$ is reached, the generation of new neighbors stops);

Let s' be the best neighbor generated, add move $s \rightarrow s'$ into T and remove the oldest tabu from T (i.e., update tabu list T);

s:=s'; nbiter := nbiter + 1;

***endwhile**;*

Output:

If f(s)=0, a coloring of G with k colors is obtained, otherwise no coloring has been found with k colors.

Table 3.1 The TABUCOL Algorithm

The function $A(z)$ is defined as the aspiration level of the objective function value next to be reached when the current value is $z=f(s)$. It is used like this: if a move to a neighbor s' is tabu but $f(s') \leq A(z)$, then the tabu status of this move is dropped (*i.e.*, taken away from the tabu list), which means this move is considered as a normal move because its quality is good enough to ignore its tabu status. $A(z)$ is set to $z-1$ initially. Then whenever an s' with $f(s') \leq A(f(s))$ is obtained, the aspiration level $A(f(s))$ is set to $f(s') - 1$, *i.e.*, $A(f(s)) = f(s') - 1$. The complete algorithm called TABUCOL is given in Table 3.1.

In the paper "*Tabu Search for Large Scale Timetabling*" [Hertz91], Hertz extended TABUCOL to both course and examination timetabling problems. An application is the scheduling of 1851 exams (including oral and written exams) in a period of five weeks for the Swiss Federal Institute of Technology in Zurich. There are 4929 students taking these exams and 60 periods (two periods in each day) available. The assignment was desired to meet the constraints for each student (1) no more than one exam is scheduled in same period (*i.e.*, the first order conflict); (2) the first oral exam was scheduled after the last written exam; and (3) there is at least one day free between two exams. It was found that there was no feasible solution satisfying all these special requirements, so some constraints are relaxed and penalized. For example, each pair of exams (x, y) will be penalized if x and y are given on the same day or on consecutive days. Hertz claimed that compared with many different algorithms, only the Tabu Search could generate a timetable to avoid the first order conflicts. The other conflicts occurred on the average fewer than one problem of constraint type (2) and (3) for each student. The TABUCOL described above was revised to another algorithm listed in Table 3.2.

Hertz noticed that two kinds of neighborhoods can be considered. The first one moves one exam from a period to another period. The second neighborhood exchanges an exam x in a period with an exam y in another period. The second kind of exchange is especially useful when it is desired to fix the cardinality of each period.

Initialization:

s:=initial solution in X;

nbiter:=0; *(* current iteration*)*

bestiter:=0; *(* iteration when the best solution has been found *)*

bestsol:=s; *(* best solution *)*

T:=Φ; initialize the aspiration function A;

While** f(s)>f* **and** (nbiter-bestiter<nbmax) **do

nbiter := nbiter + 1;

Generate a set V of solutions s_i in N(s) which are either not tabu or such that A(f(s))≥f(s_i);*

Choose a solution s minimizing f over V*;*

Update the aspiration function A and the tabu list T;

***if** f(s*) < f(bestsol) **then** { bestsol :=s*;bestiter := nbiter;}*

s := s;*

endwhile;

Note:

X is the set of feasible solutions; *f*, the objective function; *N(s)*, the neighborhood of a solution *s* in *X*; |*T*|, the size of the tabu list; |*V**|, the number of neighbors generated at each iteration; *f**, a lower bound for the objective function *f*; *A*, the aspiration function; *nbmax*, the maximum number of iterations between two improvements of the best solution.

Table 3.2 The Revised TABUCOL Algorithm

§3.2 Boufflet and Negre

According to the paper "*Three Methods Used to Solve an Examination Timetable Problem*", they have developed and used three tools to solve a real-world problem of examination timetabling at the University of Technology of Compiègne [BoufletS96]. They tried to assign three examination sets, which have 94, 101, and

109 exams, respectively, into 20 periods. Many constraints are considered in the model , including some constraints unusual or not typical.

The Tabu Search techniques used are very similar to but simpler than Hertz's [Hertz91]. Starting from a solution, they generate its neighbors, considering all the moves of exam from one stable set to another one. All the feasible moves are generated and the one giving the best objective function is chosen.

For each iteration, the number of neighbors is equal to $n(t-1)$ (where t is the number of planning periods or timeslots and n is the number of exams to schedule.) Regarding the tabu list, only short term memory (recency-based) is applied, and the size of the tabu list is set to 7 (same as Hertz's).

The three tools are based on a tree search, the Tabu Search technique, and an interactive computer aided design system, respectively. The initial solution is provided by a heuristic based on the tree search method, but no backtracking is applied. The initial solution may use more periods than 20, i.e., if they can not use one of the first 20 colors to schedule an examination set, they create a new planning period. Then the exams which have a color greater than 20 are distributed to the basic planning periods in order to minimize the number of violations of period constraints, and this constraint is fully respected at the beginning of the algorithm. Applying the Tabu Search, the changes made to the solution will improve the two main constraints at the expense of the planning period constraint.

They found the first method, a tree search, is the most effective among the three methods, but the tabu technique may be a convenient alternative because of higher solution quality.

§3.3 Carter's Benchmark

Strictly speaking, Carter *et al* [CarterLL96] neither present the Tabu Search strategies to solve the examination timetabling problems, nor compare Tabu Search

strategies with their strategies based on four sorting algorithms (the largest degree, saturation degree, largest weight degree, and largest enrollment) in their article "*Examination Timetabling Algorithmic Strategies and Applications*". But they gave an evaluation model for the examination timetabling problem, that induces a "cost per student" concept and considers no side constraints.

For a given set of exams to be scheduled and the number of time slots, they construct a conflict-free examination timetable (*i.e.*, no student is asked to take two exams at a time) while spreading student exams as evenly as possible over the schedule. In order to avoid too many secondary parameters to affect the results and conclusions, they do not consider any side constraints. In other words, the problem is modeled as an unconstrained problem. For example, there are no constraints for the maximum number of seats, no constraints for the maximum number of exams per timeslot, and no pre-assigned exams.

The objective function, or the cost function, is expressed as below:

$$f(x) = 16W_1 + 8W_2 + 4W_3 + 2W_4 + W_5$$

where W_i is defined as the number of students who have to take two exams, i time slots apart. For instance, W_1 is the number of students taking two consecutive exams.

In this model, the first order conflicts are modeled as hard constraints. The second order conflicts are treated as soft constraints, and included into the objective function f that measures the quality of the solution. A measurement factor called *cost* is introduced to measure and assess the quality of potential schedules. Assuming that consecutive time slots lie one unit apart, we define f as assigning a proximity cost w_i whenever a student has to take two exams scheduled with i time slots. The cost of each conflict is thus multiplied by the number of students involved in both exams. The cost decreases from 16 to 1 as follows: $w_1=16$, $w_2=8$, $w_3=4$, $w_4=2$, and $w_5=1$. The *cost* function is then normalized based on the total number of students, *i.e.*, the average "cost per student" is used to evaluate the quality of a solution instead of the cost function value.

They provide 13 sets of real data for test, which are downloadable (the URL is ftp://ie.utoronto.ca/mwc/testprob), and present the results of forty different strategies tested. Since this test bench was established, Di Gaspero and Schaerf have applied Tabu Search techniques on them and published their results [GasperoS00]. Burke *et al* have done some test with the Memetic algorithm (a combination of an evolutionary algorithm and local search) with those test data [BurkeNW96].

§3.4 Di Gaspero and Schaerf

The paper "*Tabu Search Techniques for Examination Timetabling*" [GasperoS00], describes an ongoing research on the development of a solution algorithm for examination timetabling based on Tabu Search (TS). In fact, their TS algorithm is very similar to the TS algorithm for graph coloring proposed by Hertz and de Werra. The search space is composed of all complete colorings of the graph, including feasible ones. The cost definition is same as that used in [CarterLL96].

Regarding the neighborhood definition, the authors consider two states as neighbors if they differ for the coloring of a single node. Therefore, a move corresponds to changing the color of one node, and it is identified by a triple (*node, old_color, new_color*). Regarding the notion of inverse of a move, the authors found, with many experiments, the one that has given the best results is the one that considers the inverse of $<u, c_1, c_2>$ to be any move of the form $<u, -, ->$. That is, the color of the node cannot be changed again to any new one.

The authors also found they can benefit from the two violation lists used in their algorithm. The first list is called violation list *VL*, which contains the nodes that are involved in at least one violation (either first or second order). A second (shorter) list *HLV* contains only the nodes that are involved in the first-order violations. In different stages of the search, nodes are selected by an exhaustive exploration of either *VL* or *HLV*. Namely, it selects from *HLV* when there are some hard violations,

and resorts to *VL* in any iteration in which *HLV* is empty. Note that nodes not in the lists are never analyzed.

The model adapted in their research is very simplified one, such constraints as room capacities, preferences, and pre-assignments have not been integrated into the model.

Chapter 4

Examination Timetable Modeling

In this chapter, we introduce a commonly used model for solving the examination timetabling problem with Tabu Search techniques [HertzW87] [Hertz91] [BoufflетS96] [GasperoS00]. We first discuss a graph representation of the examination set by which the conflict relations between examinations are reflected very simply, and then we apply the graph vertex-coloring concept to the scheduling of examination timetables. Based on this model, several considerations of typical issues such as the types of atomic moves and simplification of computation in the local search, weighting of conflicts between timeslots, pre-assigned examinations, and the pre-assigned classroom issue are presented as well.

§4.1 Graph Representation of Examination Timetable

We use an undirected graph $G = G(V, E)$ to describe the examination set to be scheduled and the relations among those examinations, where, V is the set of vertices or nodes (these two terms are equivalent in this paper) and E, the set of edges in G. Each exam is represented by a vertex v_i. An edge connects two vertices if the corresponding exams have at least one student in common. An example is given in Figure 4.1.

Let V be an examination set, $V = \{v_1, v_2, \ldots, v_v\}$, or $V = \{v_i \mid 1 \le i \le v\}$, v_i be each individual exam to be scheduled, and v be the total number of examinations (denoted by $v = |V|$). We use s_i to denote the number of students (or enrollment) taking exam v_i. If there exists at least one student taking both exams v_i and v_j, we

call v_j a *conflict exam* of v_i, and vice versa. We use CV_i to represent the set of conflict exams of v_i, *e.g.*, $CV_2 = \{v_5, v_6\}$, in Figure 4.1.

If v_i and v_j are conflict exams with m students in common (*i.e.*, if m students take both exams v_i and v_j), we consider that there is an edge (v_i, v_j) in G with weight m, denoted by $w_{i,j}$, and we have $w_{i,j} = w_{j,i} = m$. Otherwise, if there are no students taking both v_i and v_j, we have $w_{i,j} = 0$ (or $w_{j,i} = 0$).

We define the *degree of exam* v_i to be the number of exams in the conflict exam set CV_i of v_i, denoted by d_i and $d_i = |CV_i|$. We define the *weight of exam* v_i to be the sum of weights of edges connected with exam v_i, denoted by w_i, and we have $w_i = \Sigma\, w_{i,j}$, for all $v_j \in CV_i$.

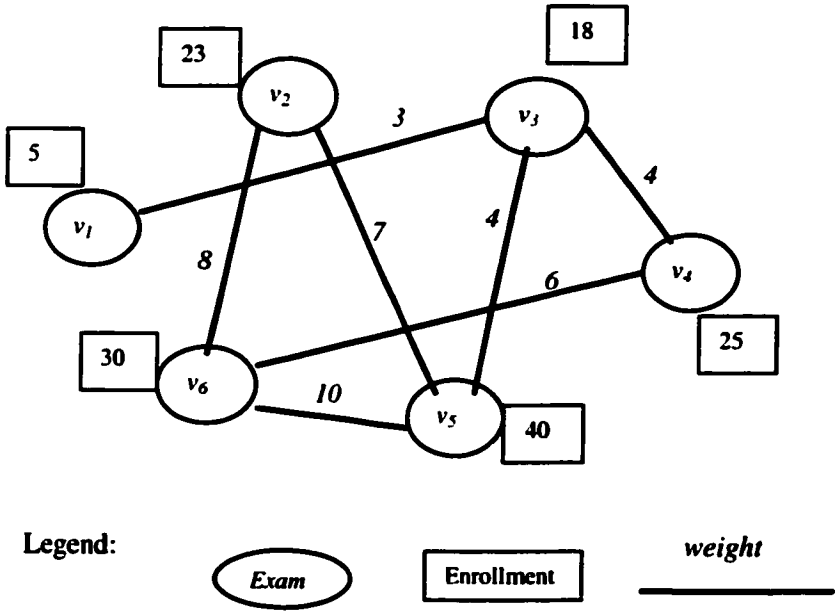


Figure 4.1. Graph Representation of Examinations

Let E be the total set of edges in G and e be the total number of edges in the graph and w be the sum of the weights of edges in E. We define the *matrix density* (denoted by *dom*) to be the ratio of the number of edges (denoted by e) to the maximum number of edges in graph G, $dom = 2e/v(v-1)$. Generally speaking, for examination timetables, the larger the density of matrix, the more difficult the optimization of examination timetable.

Exam	*Enrollment*	*Degree*	*Weight*	*Conflict Matrix*					
v_I	s_I	d_i	w_i	*1*	*2*	*3*	*4*	*5*	*6*
1	5	1	3	0	0	3	0	0	0
2	23	2	15	0	0	0	0	7	8
3	18	3	11	3	0	0	4	4	0
4	25	2	10	0	0	4	0	0	6
5	40	3	21	0	7	4	0	0	10
6	30	3	24	0	8	0	6	10	0
Total	141	e=7	w=42	*dom* = 0.47					

Table 4.1. Tabular Representation of Examinations

§4.2 Graph Coloring and Examination Scheduling

Let T be a given set of consecutive time slots $T = (T_1, T_2, \ldots, T_k)$, where k is the number of timeslots. In order to construct an examination timetable, we must assign each exam into only one timeslot. For instance, if exam y is assigned into time slot T_a, we have $y \in T_a$ and $y \notin T_b$ for all T_b $(T_b \neq T_a)$.

Obviously, we have $\cup T_i = V$, $\cap T_i = \Phi$, and $v = \Sigma |T_i|$, which means that, for a timetable, no exam is assigned twice or more, and a time slot may be empty.

Figure 4.2. Consecutive Timeslots

For two timeslots T_i and T_j ($j \geq i$), the distance between them is defined as $dis = j-i$. *e.g.*, the distance of two adjacent timeslots is 1, and the distance of T_3 and T_1 is 2.

Let $E_{0,i}$ be the set of edges having both endpoints in T_i ($1 \leq i \leq k$), *i.e.*, the set of internal edges in T_i, which means the set of conflict exam pairs. Let W_0 be the sum of weights of edges in edge set E_0, where $E_0 = \Sigma E_{0,i}$ ($i = 1, 2, ..., k$).

Then, we examine other conflict edges between different timeslots. The first case is the conflict between adjacent timeslots (*i.e.*, consecutive timeslots, their distance is 1). Let $E_{1,i}$ be the set of edges between T_i and T_{i+1} ($1 \leq i \leq k-1$). W_1 is the sum of weights of edges in edge set E_1, where $E_1 = \Sigma E_{1,i}$ ($i = 1, 2, ..., k-1$).

In order to generalize the notation, we define $E_{n,i}$ to be the set of edges between T_i and T_{i+n} ($1 \leq i \leq k-n$, and $0 \leq n \leq k$), and W_n be the sum of weights of the edges in edge set E_n, where $E_n = \Sigma E_{n,i}$ ($i = 1, 2, ..., k-n$).

It is always a good approach to use examples to explain a complicated notation system. Assume we assign these six exams (shown in Figure 4.1) into three timeslots T_1, T_2 and T_3 in this way:

$T_1 = \{v_1, v_2, v_6\}$, $T_2 = \{v_5\}$, and $T_3 = \{v_3, v_6\}$, as shown in Figure 4.3.

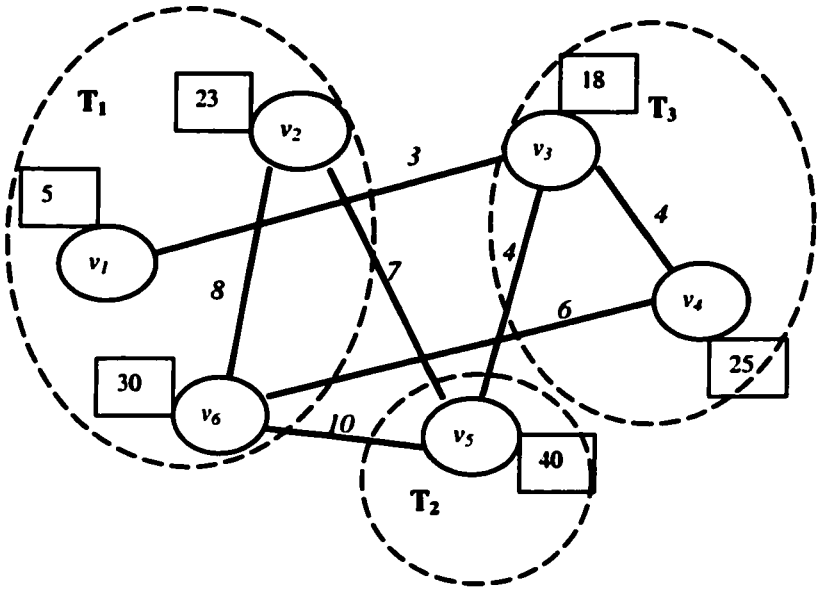


Figure 4.3. Six Exams assigned into Three Consecutive Timeslots T_1, T_2, and T_3

Timeslot	*Exams*	E_0	E_1	E_2
T_1	$\{v_1, v_2, v_6\}$	(v_2, v_6):8	(v_2, v_5):7, (v_5, v_6):10	(v_1, v_3):3, (v_4, v_6):6
T_2	$\{v_5\}$	-	(v_3, v_5):4	-
T_3	$\{v_3, v_6\}$	(v_3, v_4):4	-	-
		$W_0 = 12$	$W_1 = 21$	$W_2 = 9$

Table 4.2. Conflict Relation and W values in Figure 4.3

In this solution, we conclude very easily that there are 12 students having to take two simultaneous exams, 21 students taking consecutive exams, and 9 students to

take two exams with a session length break. Obviously, this is an unacceptable timetable, and we need to get a feasible and optimized one.

§4.3 Objective Functions and Their Optimization

For a timetable to be feasible, the sum of weights of edges having both endpoints in the same timeslot must be zero, *i.e.*, $E_{0,i} = \Phi$ and $W_0 = 0$, which means no student is required to write two or more exams at same time. In addition the sum of weights of edges between the timeslots with a given distance should be minimized. Such a timetable is called optimal. We must, of course, ensure that no timeslot requires more examination seats than are available. In other words, in order to get a feasible solution (*i.e.* a conflict-free timetable), we can construct an objective function $f(x) = W_0(x)$, then optimize it to get a solution s such that

$$f(s) = W_0(s) = 0.$$

In fact, we have to consider more constraints such as the maximum number of seats available and the maximum number of exams acceptable for a given timeslot in a real examination timetabling problem. We define $maxSeats_i$ to be the maximum number of seats available and $maxRooms_i$ to be the maximum number of rooms available for T_i.

To explain the objective function integrated with different weights on the different type of conflicts without losing generality, we consider a case which optimizes a timetable to get a feasible solution while reducing the conflicts between whose timeslots whose distance is less than or equal to 5.

We define the objective function $f(x)$ for solution x as:

$$f(x) = p_0 \times W_0 + p_1 \times W_1 + p_2 \times W_2 + p_3 \times W_3 + p_4 \times W_4 + p_5 \times W_5$$

where a solution x is subject to the conditions that:

- For any T_i, the number of enrollments is not greater than $maxSeats_i$, the number of exams is not greater than $maxRooms_i$. Note that we do not consider the case that some schools may arrange several small exams into a big classroom at same period.

- p_0, p_1, p_2, p_3, p_4, and p_5 are the penalties chosen to weight the different conflict classes in s that are characterized by W_0 , W_1, W_2, W_3, W_4, and W_5, respectively.

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

§4.4 Initial Solution and Objective Function

In order to start a tabu search, we need an initial solution, which can be generated in different ways. The initial solution may be a random one, a designated one, one optimized by other algorithms, or one obtained from the last round of tabu search.

One thing worth mentioning is the impact of initial solution on the quality of final solution. Some researchers have asserted that the initial ordering of examinations has virtually no impact on the final outcome. But others have found the opposite: the initial ordering strategy can be critical in graph coloring [CarterLL96].

§4.5 Atomic Move

An atomic move is obtained by swapping colours of an exam v_a from T_i to T_j. It can be defined as below:

Remove an exam $v_a \in T_i$, then add it into T_j ($T_i \neq T_j$), *i.e.*,

$T_i = T_i \setminus \{v_a\}$ and $T_j = T_j \cup \{v_a\}$.

Obviously, we can generate $v(k-1)$ possible atomic moves starting from an initial solution, in other words, we can generate $v(k-1)$ new solutions by atomic moves from any solution. We use an atomic move table (AM Table) to record these atomic moves.

§4.6 The Computation of Objective Function Value

In order to reduce the complexity of computation, we use the following algorithm to calculate the objective function values of every new solution.

We calculate the objective function value of an initial solution, denoted by f_0. When a new current solution s_1 is obtained from this initial solution by moving an exam x from timeslot *slot-src* to *slot-des*, denoted by a tuple (x, *slot-src*, *slot-des*), we only need to recalculate the conflicts value of several timeslots relevant to timeslot *slot-src* and *slot-des* due to the move since most of timeslots keep their conflict value (or pattern) unchanged. Assume the change of the objective value is δ_1, then the value of the objective function of solution s_1 is $f_1 = f(s_1) = f_0 - \delta_1$. For the same reason, for the i-th solution (*i.e.*, the solution obtained after i iterations), we have $f_i = f_{i-1} - \delta_i = f_0 - \Sigma\delta_k$, $k = 1, \ldots, i$.

§4.7 An Optimized Solution

A possible solution is listed in Table 4.3. We found this is a fairly good solution. No student has to take two simultaneous exams, 15 students must take consecutive exams, and 27 students must take two exams with an one session length break.

Timeslot	*Exams*	E_0	E_1	E_2
T_1	{ v_3, v_6}	-	(v_2, v_6):8	(v_1, v_3):3, (v_3, v_4):4, (v_3, v_5):4, (v_4, v_6):6, (v_5, v_6):10
T_2	{ v_2}	-	(v_2, v_5):7	-
T_3	{ v_1, v_4, v_5}	-	-	-
		$W_0 = 0$	$W_1 = 15$	$W_2 = 27$

Table 4.3. An Optimized Solution

§4.8 Distancing of the Real Relationship between Timeslots

As Burke indicated [BurkeEF98], there are a vast number of possible constraints on the timetable produced, over and above those entirely due to resources. These relate directly to the quality of the timetable in term of its usability and requirements on those who are subject to it. Some may prefer to take two consecutive exams with an overnight sleep rather than two day-time exams, a morning exam and an evening exam.

Assume we have timeslots shown in table 4.4.

A special timeslot TF can be used to reshape the relationship between real timeslots. For this fake timeslot TF, the maxSeats is set to zero, which means, in the process of local search, no student would be assigned into it. For example, we may accept this timeslot pattern:

T1M – T2A – T3E – TF – T4M – T5A - T6E – TF – TF – TF – T7M – T8A

Timeslot	Specification
T1M	Friday, 8:30-11:30am
T2A	Friday, 1:30-4:30pm
T3E	Friday, 6:00-9:00pm
T4M	Saturday, 8:30-11:30am
T5A	Saturday, 1:30-4:30pm
T6E	Saturday, 6:00-9:00pm
T7M	Monday, 8:30-11:30am
T8A	Monday, 1:30-4:30pm

Table 4.4 The Relation of Timeslots: Variants

A TF is inserted between the timeslots T3E and T4M by such a consideration that the conflict between 2 day-time slots is greater than the conflict between a evening timeslot and next morning timeslot from the perspective of giving more student's rest time between two exams. For the same reason, three TFs are inserted between real timeslots T6E and T7A because a Sunday is considered.

Note that another method is to add an attribute to a timeslot in order to store the information about the distance information between relevant timeslots. We believe that this method has more flexibility for us to solve a real examination scheduling problem from the point view of algorithm implementation.

§4.9 Considerations on Pre-assigned Examinations

In an implementation, the pre-assigned examinations can be treated as some special exams which can not be moved during the local search process once they are allocated into preset timeslots. But, we should avoid too many pre-assigned exams in a timetable because they usually deteriorate the quality of an examination timetable. The pre-assigned exams can be classified as having different privileges. We can generate two solutions, one accepts all pre-assigned exams, and one disables those pre-assigned exams with lower privileges. By comparing the qualities (*i.e.*, the objective function values) of two solutions, we may easily decide how to deal with the pre-assigned examinations in each individual timetable.

Chapter 5
The OTTABU Algorithm

In order to benefit more from the attractive but complicated Tabu search techniques, we have implemented a Tabu search algorithm, OTTABU, in which both short term memory and longer term memory mechanisms function. The Tabu search notions, the construction and operations of two types of tabu lists, the local search strategies, and the Tabu relaxation technique are presented in this chapter. The details of several routines of OTTABU algorithm are introduced here as well.

§5.1 Atomic Move, Neighborhood, and Local Search

Let s be a solution, $s = (T_1, T_2, \ldots, T_k)$, where T_i is the i-th timeslot and k is the total 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 neighbor solution (or neighbor for short) of s and all the neighbors generated from s by an atomic move as the neighborhood of s, denoted by $N(s)$.

The size of the neighborhood depends on both 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. These lists are used to generate the neighbors 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_{dis}. Note that these notations are described in Chapter 4.

Starting from an initial solution, the Tabu search algorithm iteratively explores a subset $N^*(s)$ of the neighborhood $N(s)$ 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)*.

§5.2 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 current solution *s**, the tuple *(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. 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*.

There are two methods to set the tabu tenure. One is to set a fixed tenure for all the tabu moves, another one is to assign a random tenure within a given range, for each individual move. We have investigated situations where the tenure of *TS* was set to various fixed values between 5 and 40. We also have done experiments to test how well the randomized tenure works. It is hard to conclude whether we can benefit from randomized tenure rather than fixed tenure. But, when the transition frequency based tabu list is used, the exact value of tenure for the recency-based tabu list was found to be unimportant in determining the final result and was not investigated further. But we noticed that the randomized tenure can change the search space path, which may diversify the solution space.

§5.3 Transitional Frequency Based Longer-term Memory

In order to investigate the effect of a move frequency-based longer 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 from one timeslot to another, 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 to help prevent cycling. If a node x has been moved more than 2 times and *TL* is not full, it will be put into *TL*. If *TL* is full and if some node y already in *TL* has a lower move frequency than x, we will 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 analyze 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 enrollment, degree, or weight as a light node (exam). Similarly a node (exam) with higher enrollment, 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 but most influential if moved.

In the process of search, the light nodes can act as "crack fillers" 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 Tabu search. 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 which have a high likelihood to be chosen to generate a new solution will help us to decide what is the appropriate length for *TL*, denoted as l_{TL}.

§5.4 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 neighborhood 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 $3l_{TL}$. If the search has passed $2l_{TL}$ iterations since the latest best solution was found and *TL* is full, or the current solution is greater than a threshold value, the entries in *TS* and *TL* will be dropped automatically. This threshold value changes when a new best solution is obtained and calculated by the formula $f_{best} \times r$, where r is a predefined ratio r ($r > 1$). That is, if $f(s)$ is greater than $f_{best} \times r$, where $f(s)$ is the value of the objective function of the current solution s and f_{best} is the value of the objective function of the latest best solution, both tabu lists are emptied.

Experiments show that the threshold r can be set to a value between 1.05 and 1.25. 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 $2l_{TL}$ and *TL* is full, the relaxation will lead to a dramatic downhill movement, which may in turn lead to new regions or search spaces.

§5.5 Move Preference: to Increase Diversification

Several strategies are used in the local search for diversifying the search space. The first strategy is called move preference. If we can move either of two exams that would generate the same objective function value, we choose the one with bigger enrollment. If their enrollments are equal, we choose the one with bigger degree. If they are still the same, then choose the one with bigger weight. If they are all the same for two exams, we choose the first occurrence. The theory behind this strategy is the move of the bigger exam has more influence on the solution structure. Another strategy is the randomized tenure for the short term memory tabu list. We assign a random tenure with a range of 15 to 25 for each item in the short term tabu list *TS*.

§5.6 Intensification

For a large scale examination timetable, the neighborhood $N(s)$ of the current solution s generated with nodes in V has a much bigger size than the neighborhood $N^*(s)$ 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 bigger *nbmax* chosen for the search over V^*. It is shown that the multi-pass search strategy presented below can generate better solutions and save search time, and the swap of neighborhood types works very well.

Pass 1

We start the Tabu search from a given initial solution. We generate neighbors of the current solution s with the nodes in V^* and set a bigger value for *nbmax* to allow the search to do more downhill and/or 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 to 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 neighborhood 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.

If required, we can repeat the two passes and also can set the *nbmax* to different values. 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.

§5.7 Details of the OTTABU Algorithm

As described above, OTTABU algorithm calls Tabu search routine several times by setting different parameters such as the neighborhood type (*V* or *V**) and *nbmax*. The neighborhood type in Pass 1 may be set to either *N*(s)* or *N(s)*, but it should be different from the one used in Pass 2. If the neighborhood type is *N(s)* in a pass, the *nbmax* should not be too big, otherwise, the search will be too lengthy. The initial size of the tabu list *TL* is set by estimating the exam data. An estimation method called *accumulative percentage method* will be introduced in Chapter 6.

Multi-pass search

A procedure implementing multi pass search is given in Figure 5.1. Note that the last solution obtained in the last pass is used as the initial solution in the current pass.

```
1.  s:= an initial solution s0;
2.  set penalty array p[dis];
3.  estimate np;
4.  bestsol := s; fmin := 0;  neighbortype := V* ;
5.  t:=0;
6.  while t < numPass do {
7.  s := TabuSearch(s);
8.  if neighbortype = V * then  neighbortype := V;
9.  else  neighbortype := V * ;
10. t:= t+1;
11. } endwhile;
12. output (s); exit;
```

Figure 5.1 Multi-pass Procedure

A Tabu search routine

The Tabu search routine is given in Figure 5.2. The move preference, the trigger of tabu relaxation, transition frequency table and the adaptive tabu list *TL* are shown in this routine.

Several notes on some tabu list operation related functions

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

(1) Update *TS(x, i, j)*. This function does following: all tenures are decreased by 1, drop the entries whose tenures are 0, and add *(x, i)* with a random tenure. If *s** is the best solution so far, drop *(x, j)* if it exists;

(2) Update *TL(x)*. This function does following: add (x) into the tabu list *TL* when *TL* is not full and MFT[x] ≥ 2, or replace *y* in *TL* with *x* if *TL* is full and *MFT[y] < MFT[x]*; and

(3) Update *MFT(x)*. The transition frequency will be incremented when an exam is moved to get a new solution, *i.e.*, *MFT[x]:= MFT[x] + 1;*

OTTABU does a tabu relaxation (*i.e.*, *optiter* is reset to 0, empty *TL* and *TS*) when the number of idle iterations equals to $2l_{TL}$ or the current solution is worse than *f(bestsol)*r*. *r* is a ratio greater than 1. Unlike *TL* and *TS*, *MFT* will not be reset to 0 during an OTTABU search life-time.

1. ***Routine Input**: neighborhoodtype and a solution s';*

/* Initialization */

2. *MFT[] := 0; TS:= Φ; TL:= Φ; /* two tabu lists and move frequency table */*
3. *nbiter := 0; bestiter := 0; optiter := 0;*
4. *s:= s'; f_{min} := 0; /* the low bound of the objective function */*
5. *nv := np; /* np is a value calculated by the cumulative percentage method */*
6. ***if** neighborhoodtype = V* **then** VC := V* **else** VC := V;*

/* Iterations */

7. ***while** f(bestsol) > f_{min} **and** (nbiter - bestiter) < nbmax **do** {*
8. *nbiter := nbiter + 1; optiter := optiter + 1;*

/ adaptive length of TL tabu list */*

9. ***if** |VC| <= nv **then** nv := 2|VC|/3;*
10. ***if** neighbortype = V * **then** l_{TL} := nv; nbmax := 3l_{TL} ;*
11. ***else** l_{TL} := nv/2; nbmax := 3l_{TL} /2 ;*
12. *For any x ∈ VC, generate a set of solutions NS(s) := {s_i |{s_i := s⊕m_i } from all atomic moves m_i := (x, T_i, T_j), where either both (x, j) not in TS and x not in TL or such that f(s_i)< f(bestsol); /* a tabu move accepted if it can generate the best solution */*
13. *Choose a solution s* ∈ NS(s) with minimum f over NS(s) and if more than one solution is minimal, choose one with following preferences: the lowest move frequency first, then, the largest enrollment, the largest degree, the largest weight, then the first occurrence; /* the move preference */*
14. *s:=s* ; /* new solution generated. Note that s* may be equal to, better or worse than s obtained in the last iteration */*
15. *update MFT(x); update TS(x, i, j); update TL(x); /* see the notes for details */*
16. ***if** f(s)<f(bestsol) **then** bestsol = s; bestiter := nbiter; optiter := 0;*
17. ***else if** optiter > 2l_{TL} **or** f(s) > f(bestsol) * r*
18. ***then** optiter := 0; TS := Φ ;TL:=Φ; /* Tabu relaxation */*
19. *} **endwhile**;*
20. ***return** bestsol; /* the best solution obtain so far, may be used by next pass */*

Figure 5.2. The Tabu Search Routine

Notations and Symbols

The notations and symbols used in the routines above are listed in Table 5.1.

Symbols	Specifications
bestiter	The number of iterations since the latest best solution was found
bestsol	The best solution so far
f(s)	The value of objective function of solution *s*: $f(s) = \Sigma p_i w_i$, $0 \le i \le dis$, where *dis* is the number of timeslots between two exams
MFT	The Node Move-Frequency table. *MFT[x]* is the move frequency of node *x*
m_l	Any feasible atomic move (x, T_i, T_j) that moves a node x from T_i into T_j $(j \neq i)$
nbiter	The current iteration number
nbmax	The max number of null iterations
np	The estimated number of most active nodes in the examination set
N(s), N(s)*	*N(s)* all neighborhood solutions of solution *s*; *N*(s)* is a subset of *N(s)*
NS(s)	*Either N(s)* or *N*(s)* of solution *s*
optiter	The number of iterations since the latest tabu relaxation
r	A threshold triggering a tabu relaxation, *i.e,.* a ratio used to limit the uphill movement during the Tabu search
s	The current solution
*s**	The minimal solution generated in some iteration from the current solution *s*
T_i	Timeslot *i* (or period *i*)
TL, l_{TL}	The longer term tabu list and its max length
TS, l_{TS}	The short term tabu list and its max tenure generated randomly
V	All nodes in graph, *i.e,.* all examinations
*V**	A set of nodes having at least one conflict in current solution *s*

Table 5.1 The Notations and Symbols in the Procedures

Chapter 6

Quantitative Analysis of the Examination Graph

It is found helpful to investigate the internal attributes of an examination set. One of them is the number of the lightest exams. We use the term, the *lightest nodes*, to refer to those exams which are much smaller than the average exam in the enrollments, degrees and weights. Because recency based tabu search techniques use one (or more than one) tabu list with a limited tabu tenure to prevent the back and forth movement, the number of lightest nodes is critical to the efficiency of tabu searching.

In order to choose an appropriate parameter for the length of the second tabu list *TL*, we need to analyze the graph of the examination set.

Assume there are v exams (or nodes) and e edges in an exam graph $G=(V, E)$. We define the *enrollment* s_i of a node v_i to be the number of students taking this exam; the *degree* d_i of a node v_i to be the number of edges connected with v_i directly, and the *weight* w_i of the node v_i to be the sum of weights of these edges.

We calculate some metrics as below:

- The total number of edges, $e = \Sigma d_i/2 \ \ (i = 1, 2, \ldots, v)$
- The total enrollment, $s = \Sigma s_i \ \ (i = 1, 2, \ldots, v)$
- The total weights of graph, $w = \Sigma w_i/2 \ \ (i = 1, 2, \ldots, v)$
- The density of matrix, $dom = 2e/v(v - 1)$
- The average enrollment per exam, $ae = s/v$
- The average degree per exam, $ad = 2e/v$

- The average weight per exam, $aw = 2w/v$
- The average exams per student, $ee = s/v$

If the node set $(v_1, v_2, ..., v_v)$ is sorted according to the number of students enrolled, by the degree of node, and by the weight of node, we can create 3 ordered (or sorted) series:

- $s' = (s'_1, s'_2, ..., s'_v)$, where $s'_i \leq s'_j$ for any $i < j$.
- $d' = (d'_1, d'_2, ..., d'_v)$, where $d'_i \leq d'_j$ for any $i < j$.
- $w' = (w'_1, w'_2, ..., w'_v)$, where $w'_i \leq w'_j$ for any $i < j$.

The Table 6.1 is the table showing several very important metrics of three real examination sets (see Appendix A for more details).

University and Data Set	OTT-F-96	CAR-F-92	UTA-S-92
Total exams	771	534	622
Total students	14,032	18,419	21,266
Total enrollments	46,899	55,522	58,979
Total degrees	18,522	20,305	24,249
Total weights	140,704	151,000	152,202
Average students per exam	60	102	94
Average degrees per exam	48	74	77
Average weights per exam	182	278	244
Average exams per student	3.34	3.01	2.77
Density of matrix	0.062	0.138	0.126

Table 6.1 Some Important Metrics of Examination Set

§6.1 The Lightest Nodes Are the Likely Most Active Nodes

If we look into the sorted data, we find some exams are very small relative to the large ones. Some courses are taken by few students with few conflicts or no conflict, but some are very common courses with bulky enrollments, which conflict with most of the exams. For instance, in UTA-S-92 exam set, there are twenty exams which are taken by at most five students, and conflict with at most eight other exams, in contrast to an average exam which is taken by 94 students with 77 conflicting exams. In CAR-F-92 and OTT-F-96, some exams have no conflicts with other exams.

Experiments have shown that the nodes (exams) with lower enrollment, degree, or weight have a high probability to be the most active nodes moved during the local search process. This, of course, not only helps the generation of more optimal solutions but also increases the probability of cycling in the tabu search. 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*.

The Table 6.2 gives more details about these three exam sets.

Based on the series s', d', and w', we can estimate how many light and heavy nodes there are in the graph with a method called the accumulation percentage estimation.

Exam	OTT-F-96			CAR-F-92			UTA-S-92		
	s	*d*	*w*	*s*	*d*	*w*	*s*	*d*	*w*
1	1	0	0	2	0	0	1	1	1
2	1	0	0	2	1	1	1	2	3
3	1	0	0	3	1	1	2	3	3
4	2	0	0	3	1	1	2	3	3
5	2	0	0	5	1	2	2	3	4
6	2	0	0	5	1	2	2	4	4
7	2	0	0	6	2	2	3	4	5
8	2	0	0	6	2	3	3	6	6
9	2	0	0	6	2	5	3	6	6
10	2	0	0	7	2	5	3	6	7
11	2	1	1	7	3	5	3	6	7
12	3	2	2	8	3	6	4	6	8
13	4	2	2	8	4	6	4	7	8
14	4	2	3	8	5	6	4	7	8
15	4	3	3	8	5	9	4	7	8
16	4	3	3	9	5	11	4	8	8
17	4	3	4	9	6	11	5	8	8
18	4	3	4	10	6	11	5	8	8
19	4	3	5	10	6	12	5	8	9
20	5	3	5	10	6	12	5	8	10

Table 6.2 An Examples of the Lightest Exams

§6.2 Accumulative Percentages Estimation

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

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

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

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

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

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

i.e. $ww = (ww_1, ww_2, \ldots, ww_v)$,

where $ww_j = \Sigma\ w'_i\ (i = 1, 2, \ldots, j)$, for $j = 1, 2, \ldots, v$, which indicates the accumulative weights of the j lightest exams according to their weights.

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

Accumulative Percentage	*s*	*d*	*w*	*np*	*np/v(%)*
5%	199	171	218	196	25%
10%	299	258	316	291	38%
15%	372	321	388	360	47%
50%	648	597	653	631	
85%	757	738	757	751	
90%	763	752	763	759	
95%	768	763	768	766	

Table 6.3. Accumulation Percentages (Data: University of Ottawa, Fall, 1996)

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

We notice that 196/766 or 25% of the total nodes in the graph have only 5% of the total enrollment, 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.

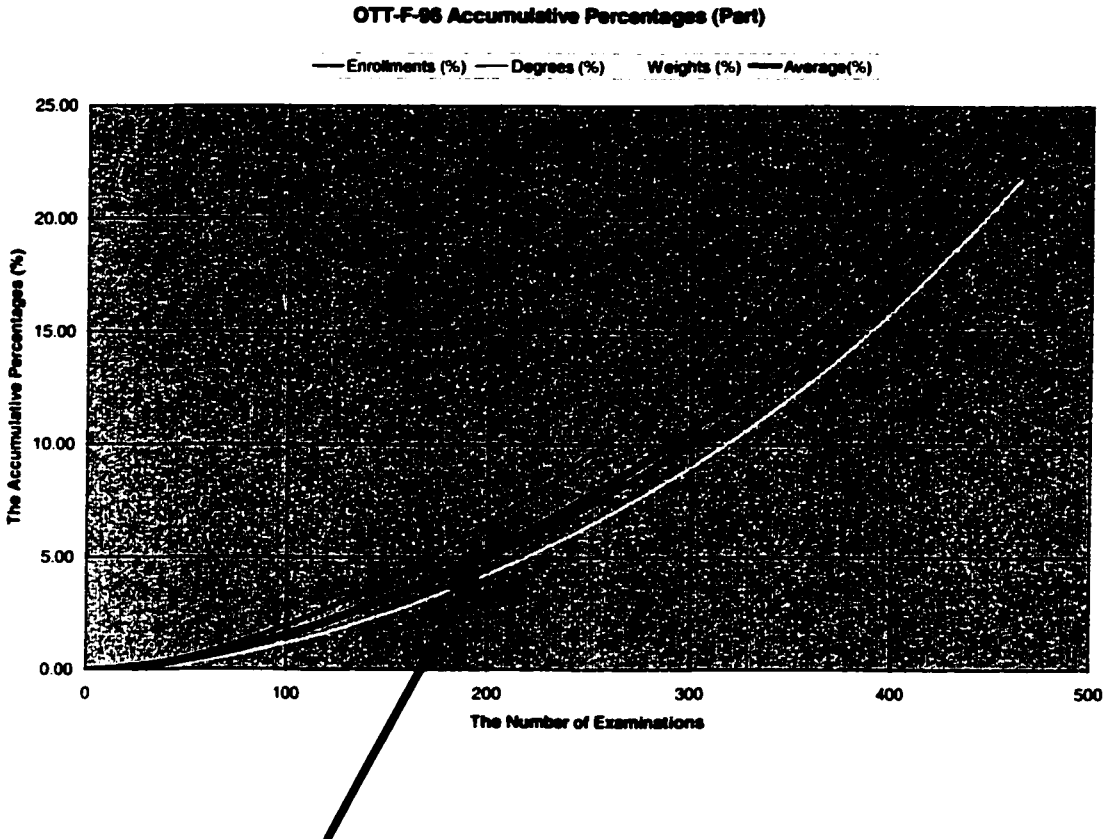
§6.3 Estimating the Length of the Longer-term Memory Tabu List

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 $A = (0, 5\%)$ are the most active nodes shown in Figure 6-1. The area $(0, 5\%)$ is the number of lightest exams

estimated at the accumulative percentage 5%. The length of the long term tabu list l_{TL} can be set to the number of elements in area A, for example, l_{TL} can be set to 196 shown in Figure 6-1.

For large examination tables, we found that this estimation method works very well to choose the appropriate values of l_{TL} and *nbmax* to make the tabu search more efficient and to get better solutions. But, for small examination tables, this estimation method does not help much. Our experiments have shown that the area $B = (0, 10\%)$ could be good for some small individual applications. More practically, if the total number of examinations is small, for example, less than 100, the area B can be chosen.

The Figure 6-2, 6-3 and 6-4 give the accumulative percentages of three real examination sets: OTT-F-96, UTA-S-92 and CAR-F-92.



(0, 5%), i.e., there are 200 lightest exams estimated by 5% accumulative.

Figure 6-1 OTT-F-96 Accumulative Percentages (Part)

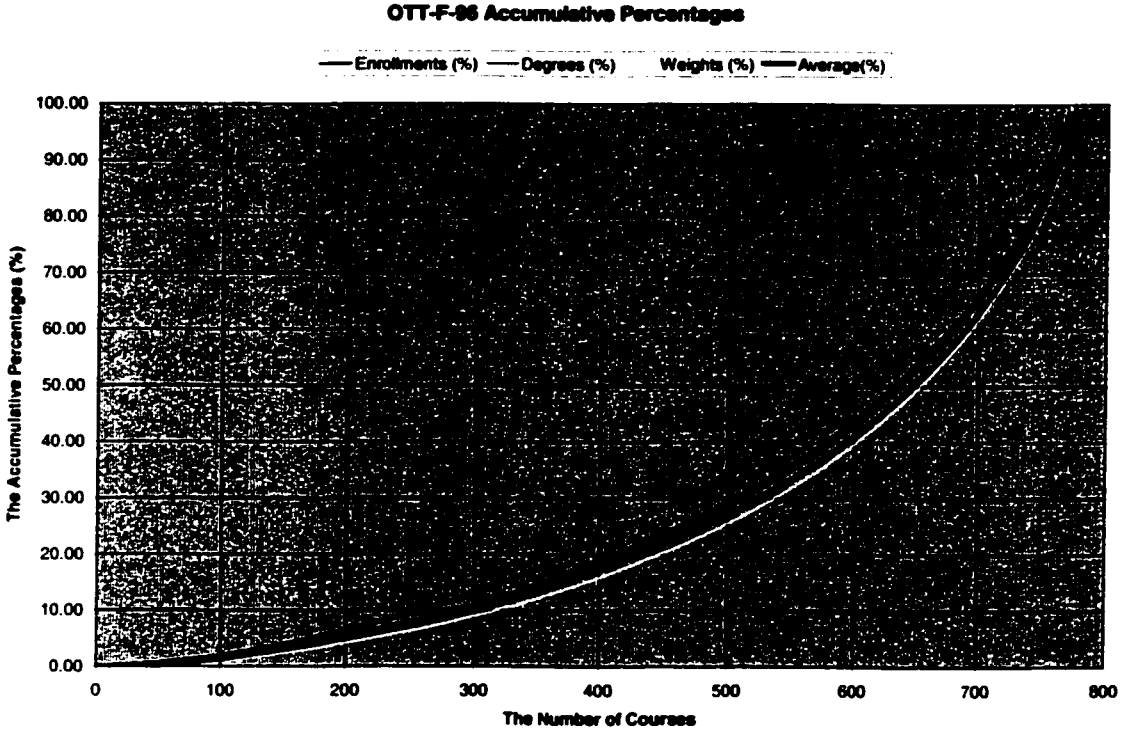


Figure 6-2 OTT-F-96 Accumulative Percentages

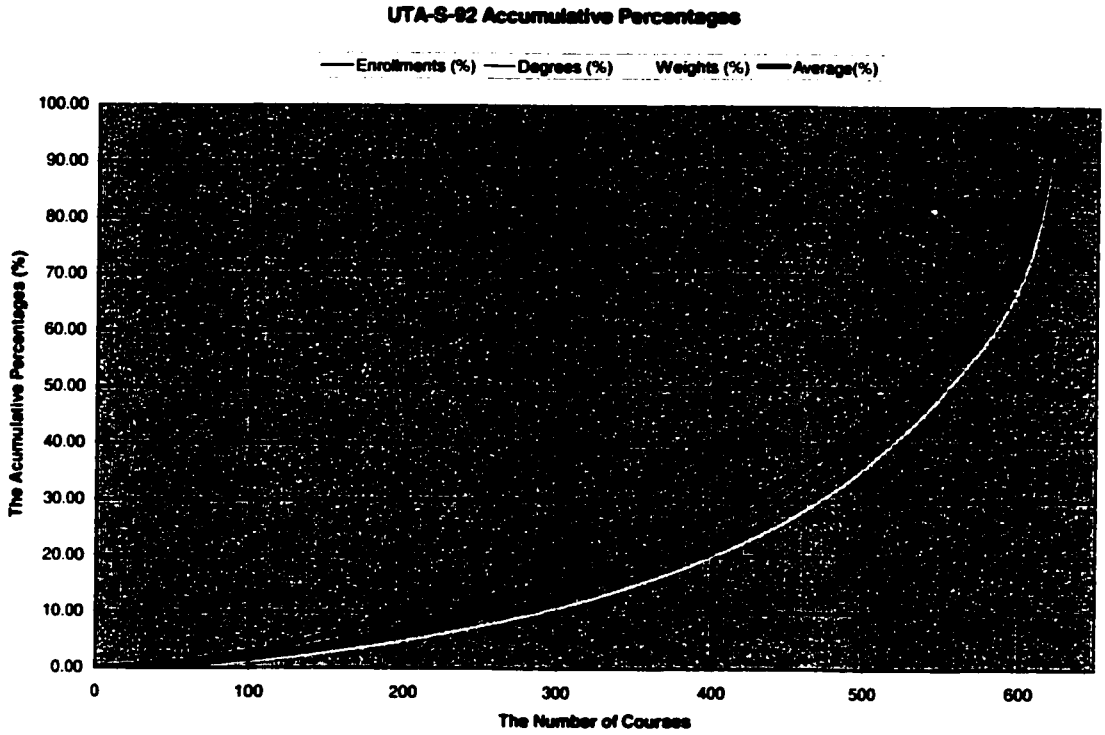


Figure 6-3 UTA-S-92 Accumulative Percentages

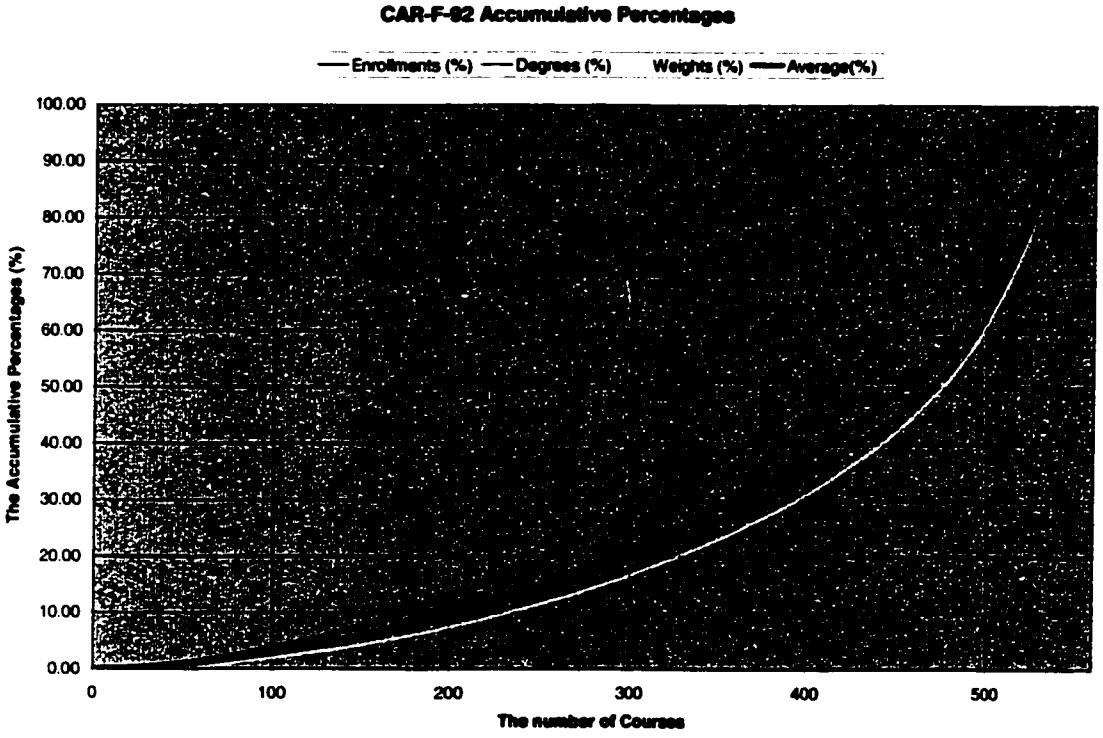


Figure 6-4CAR-F-92 Accumulative Percentages

Chapter 7

The Complete System: OTTASYS

We have constructed a multiphase system OTTASYS to heuristically optimize the solutions for the Examination Timetabling problem. As a subsystem of the OTTASYS, the OTTABU is executed multiple times with different penalty sets in the each phase of the complete system. We have done many experiments to prove that we can benefit from this system. The next chapters give the details of the experiments and results, which shows that this multi-phase system can generate much better solutions than a single phase one.

§7.1 OTTASYS -- A Multi-Phase System

A diagram is given in Figure 7.1, which describes the four-phase procedure to generate optimal solutions from a not so good initial solution.

Phase One:

- Analyze the examination set with quantitative analysis methods and estimate the number of the lightest exams;
- Apply the bin packing algorithm to generate a solution s_{bp} (with a given number of timeslots).
- If s_{bp} is a feasible solution, 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:

$f(s) = p_0 W_0, \qquad p_0 = 1.$

- 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 redo phase one until a feasible solution found, otherwise proceed to Phase Three.

Phase Three:

- Construct an objective function with such penalties that

$$f(s) = \Sigma p_i W_i, \quad p_i = 2p_{i+1}, p_{dis} = 1, 0 \le i \le dis.$$

- Apply OTTABU to restructure the initial solution s_{bp0} to get s_{inf}. Usually, s_{inf} is infeasible because the small penalty for the first order conflict will induce the solution back into an infeasible region.
- If s_{inf} is an infeasible solution (*i.e.*, there are no first order conflicts). Apply Phase Two to this solution to get a feasible solution. This solution is named as $s_{restruc0}$.
- If a feasible solution is obtained, go to Phase Four.

Phase Four:

- Use the solution generated in phase one or phase two as the initial solution. Construct an objective function with such penalties that

$$f(s) = \Sigma p_i W_i, \qquad p_0 >> 1, p_i = 2p_{i+1}, p_{dis} = 1, 1 \le i \le dis, \text{ or}$$

- Apply OTTABU to the feasible solution to optimize the solution s_{optim}.

If computation time is not critical, start from s_{optim} to generate a better one.

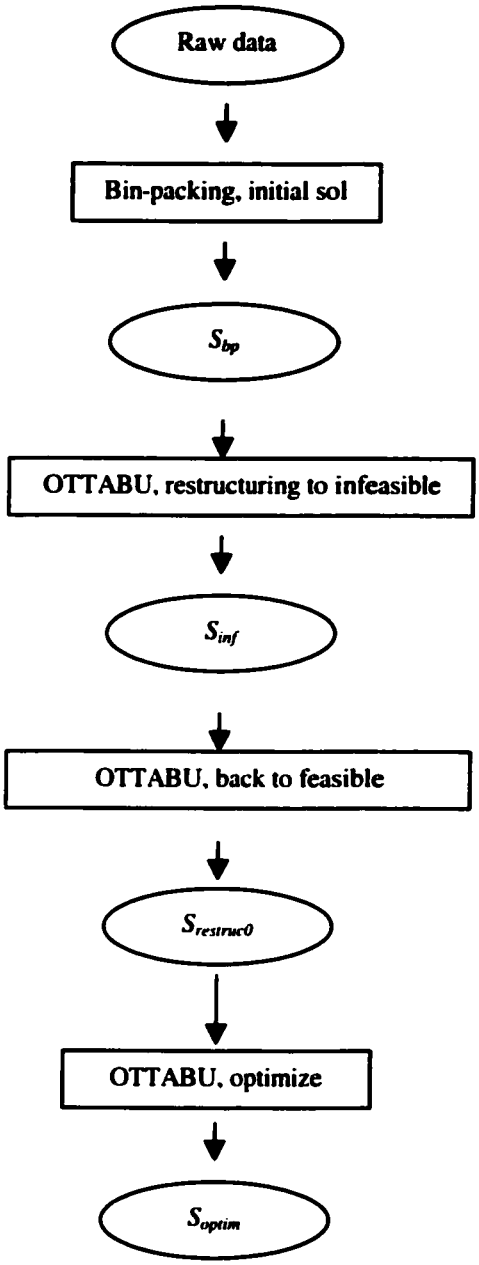


Figure 7.1 A Diagram of Solution Evolution

§7.2 Generating Initial Solutions: A Bin-packing Algorithm

An algorithm OTTAPACK derived from a generic bin packing algorithm (largest enrollment first) has been designed and used to generate an initial solution. The main idea for this algorithm is presented in next paragraphs. 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 enrollments $s_1, s_2, \ldots, s_v$ and T consists of t timeslots with capacities $c_1, c_2, \ldots, c_t$. We assign the exam with the largest enrollment 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 unsigned exams for $T_2, T_3, \ldots, T_q$. If q is equal to or less than t and all exams are assigned, we get 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. In this case, we will get an infeasible initial solution.

In case all time slots are full and there are still some exams not assigned, this trial of bin-packing should fail and more periods are needed. Even though a solution may be obtained theoretically when the total seat capacity of *all* timeslots is greater than or equal to the total enrollments, it is not difficult to conclude that, due to inevitable exam conflicts for any examination timetables, the total seat capacity would be much greater than the total enrollments for getting a real and feasible solution with minimized conflicts between consecutive timeslots.

Note 1

Theoretically, this complete system can start from any initial solution from any phase. For example, we can start from a random solution, then follow the procedure, phase by phase, to generate optimal solutions. Or, given a well-done solution, we can start Phase Four directly. But, a question is how well can we make it work if we restructure a well formed solution into an infeasible one, force it back to feasible, and then optimize it. The answer is "not very well". Experiments have shown OTTASYS can not guarantee that such a back-forth procedure will result in better and better solutions.

Note 2

OTTABU is implemented in Turbo PASCAL for Windows (TPW 1.5) and the source code is of approximate 4,000 lines. Due to the limitation of memory management in the ancient TPW version, OTTABU restricts itself to process examination timetables with at most 1000 exams. The program has been run in several PCs including my Pentium 133 Toshiba Satellite 400 Laptop and Pentium III 533 IBM PC.

Chapter 8

Experiments and Analyses

Three sets of real data, UOO-F-96, CAR-F-92, and UTA-S-92, are used in the experiment. The details of the test data are given in Appendix A. Each set of exams is scheduled into 36 timeslots and each timeslot has 2200 seats at most. In order to avoid seating large numbers of students at the end of examination sessions (where there are no following exterior edges), the seating for these three timeslots is reduced to 1100.

The objective function is constructed as

$$f(x) = p_0 W_0 + p_1 W_1$$

where the penalties are assigned to different values in the experiments below.

§8.1 Experiments and Purposes

The following experiments are designed and conducted to test how OTTASYS and its main components perform. Figure 8.1 gives a diagram showing the numbering of these experiments, their initial solutions and results obtained. The results can be found in section §8.3, §8.4, and §8.5.

General parameters and their setting

A tabu list, *TS*, implements the short-term memory or recency move. For an exam *x* and its move (*x, source-period, destination-period*), the tuple (*x, source-period*) is added to *TS* with a randomized tenure 15~25.

The system also uses a second tabu list *TL*, based on the transition frequency (or move frequency) of each exam. The size of *TL*, l_{TL}, is set to an initial value determined by the estimation method at 15% accumulative percentage and adjusted along with the size of neighborhood during the search process.

A two-pass OTTABU is used for each experiment, the neighborhood type is *N*(s)* in the first pass, and *N(s)* in the second pass.

The penalty for the first order conflict, p_0 is set to *500*, and the penalty for the second order conflict, p_1 is set to *1*. So, during the search, an infeasible solution is unlikely to be accepted because the large penalty for the first order conflict.

When the neighborhood type is *N*(s)*, the maximum number of null iterations, *nbmax*, is set to $3l_{TL}$. *nbmax* can be adjusted dynamically when l_{TL} changes to adapt to the size of the neighborhood. When only the *TL* is used, *nbmax* is set to a fixed value, which is equal or close to that value when both *TS* and *TL* are used. When the neighborhood type is *N(s)*, the *nbmax* and l_{TL} are reduced to a half in order to reduce the number of search iterations.

When the tabu relaxation mechanism is enabled, the threshold value, *r*, is set to 7% and the threshold iteration is set to a value which is equal to two-third of *nbmax*. The tabu relaxation refers to such actions that all elements in *TL* are discarded but the move frequencies remain. The relaxation is triggered when too much up-hill movement occurs or the number of null iterations consumed is equal to two-third of *nbmax*.

For the details about how these parameters are set in the algorithm, please refer to Chapter 5.

For each experiment, five instances or solutions are generated because of the randomized tabu tenures for *TS*, which could generate solutions with different qualities or values.

Initial Solution

Experiment 1 (Generating an initial solution s_{bp0})

Generate a solution s_{bp} with bin-packing algorithm. If s_{bp} is not feasible, it must be modified into a feasible one in this way: apply OTTABU to minimize W_0 to zero (with the penalty p_0 set to 1, and p_1 to 0, *i.e.*, to eliminate the first order conflicts). The feasible solution is named s_{bp0}. The results are listed in Result 1.

Optimization of s_{bp0}

The following experiments are to optimize s_{bp0} directly with different strategies:

Experiment 2 (Normal TS only)

All parameters are set as in the general case described above. Only the short term memory is used. $N^*(s)$ is used in pass 1 and $N(s)$ in pass2. The tabu relaxation is disabled. The results are listed in Result 2.

Experiment 3 (Normal TS only with a new strategy for the neighborhood type)

All parameters are the same as in Experiment 2, but the neighborhood type is $N(s)$ in pass 1, $N^*(s)$ in pass 2. This Experiment is designed to investigate the impact of neighborhood type (compared with Experiment 2). The results are listed in Result 3.

Experiment 4 (TS plus TL)

All parameters are set as the general case described above. The tabu relaxation mechanism is disabled. The results are listed in Result 4.

Experiment 5 (TS plus TL and tabu relaxation enabled)

This is the typical OTTABU method which is a combination of Experiment 4 and the tabu relaxation techniques. The results are listed in Result 5.

$s_{restruc0}$: a restructured s_{bp0}

After the optimization of solution s_{bp0}, and a feasible solution obtained, a special phase is applied to drive the feasible solution back to infeasible one, aimed at restructuring the solution structure dramatically.

Experiment 6 (Restructuring the initial solution s_{bp0})

All parameters are the same as in Experiment 2, but the penalty for the first order conflict, p_0 is set to 2. The penalty for the second order conflict, p_1 is still 1. This experiment will restructure the feasible initial solution s_{bp0} rather than optimize. Because of the small penalty for the first order conflict, we can expect that an unfeasible solution, called s_i, will be generated. If it is infeasible, it must be scheduled to get a feasible solution (called $s_{restruc0}$) with the method described in Experiment 1. The results are listed in Result 6.

Optimization of $s_{restruc0}$

Repeat the Experiment 2, 4, and 5, respectively, to optimize $s_{restruc}$. The results are listed in Result 7, Result 8, and Results 9, respectively.

Experiment 7 (TS plus TL, a larger threshold value for tabu relaxation)

This experiment is to test the impact of a larger threshold value. This is, a larger value, 15% is used in this experiment, unlike 7% in the previous experiments. The results are listed in Result 10.

Experiment 8 (Normal TS only with tabu relaxation)

All parameters are same as in Experiment 2, and the tabu relaxation mechanism is enabled. The purpose is to test the impact of tabu relaxation on the tabu list *TS*. The results are listed in Result 11.

In search of Better Solutions

Choose some optimal solutions, and continue the experiment 5 to search for better solutions. The solutions obtained in this experiment are not intended for comparison with other researchers' results because they do not have generality.

A note

The solutions in the following tables are denoted as $S_{conflic}$, where S is a feasible solution with second order conflicts, and *conflic* is the number of second order conflicts in solution S. The less the *conflic*, the higher the quality of solution S. For example, the solution S_{3500} is better than S_{3564}.

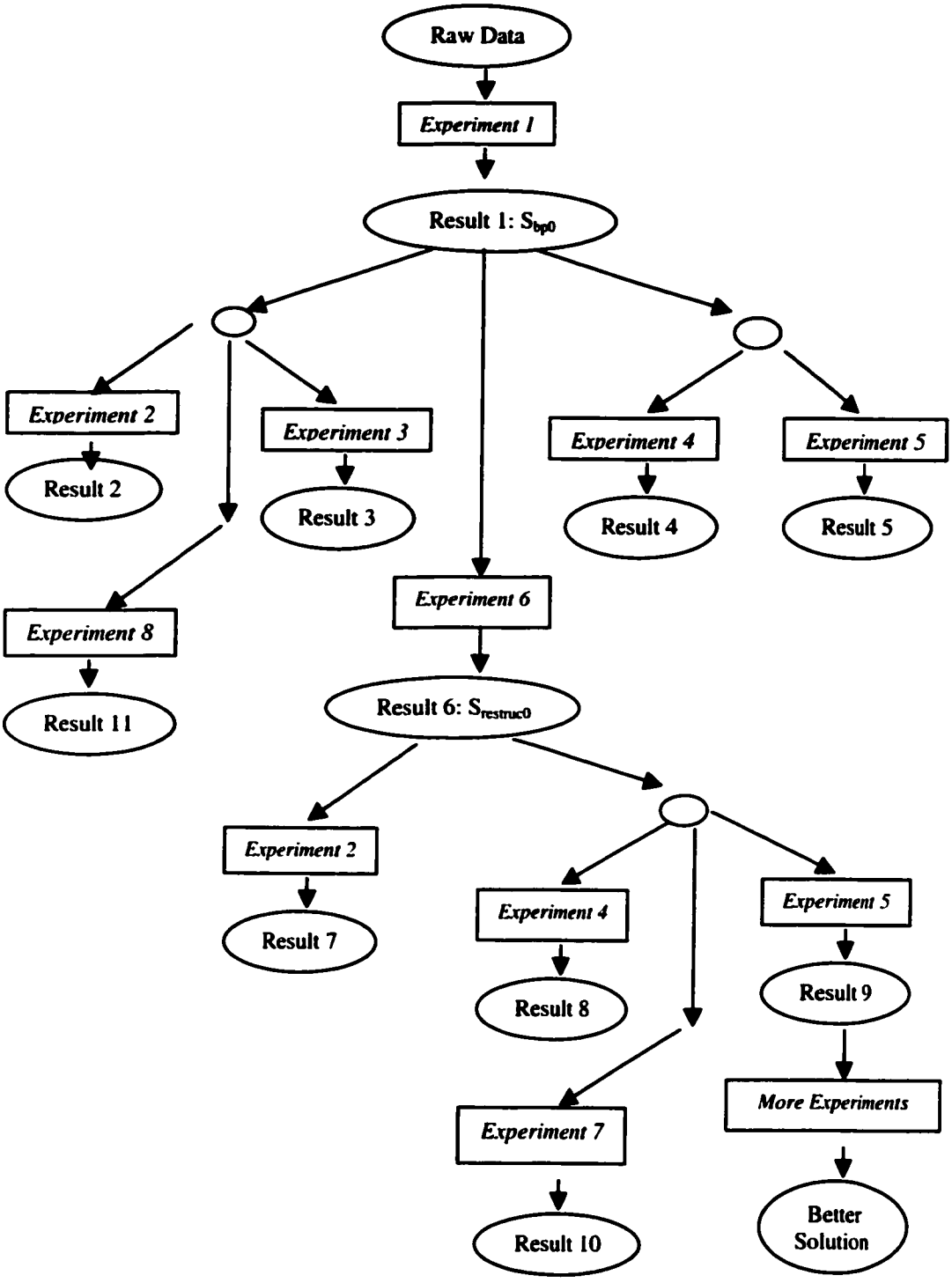


Figure 8.1 A Diagram of Experiments and Results

§8.2 Results and Analysis

We have conducted a large number of experiments and gathered the results which are classified and listed in the later sections. By analyzing the results, we found:

1. The process of restructuring an initial solution can improve the performance dramatically.

Because OTTABU could be trapped in a local optimum, the initial solution is critical to the performance of OTTABU. It seems to be true in most cases (in fact, in the three instances we tested, there is no exception). The process of restructuring a feasible solution across the feasible-infeasible border, then back to feasible worked well in the three instances. I believe this is a kind of strategic oscillation as described by Glover [GloverL97]. Even thought no solid theory explains why this strategy can improve the performance of solution searching, we would think this process of restructuring may flatten the hills and some mountains on a plateau and make it easier and likelier that the search goes down into a valley or low land from the rim of a plateau. If my hypothesis is true, after the search does go down from a plateau, it might be trapped again at some location away from the plateau it descended from.

The improvement resulting from the process of restructuring can be found in Table 8.2.1.

Note that the numbers (*n1*, *n2*) in the following tables are the number of students with the first order conflicts and the second order conflicts, e.g. (0, 5998) means that there are no student taking two exams at a time and there are 5998 students taking two consecutive exams during the 36 examination periods.

Solutions and Test Data	S_{bp0}	Average optimal solution of the Result 5 generated from S_{bp0}	$S_{restruc0}$	Average optimal solution of the Result 9 generated from $S_{restruc0}$	Improved from the result 5 to the result 9
UTA-S-92	0, 5998	0, 3557	0, 4179	0, 2811	19%
CAR-F-92	0, 8668	0, 2549	0, 6668	0, 2022	21%
OTT-F-96	0, 8686	0, 555	0, 1173	0, 512	18%

Table 8.2.1 Comparisons of solutions generated from s_{bp0} and $s_{resctruc0}$

2. OTTABU can generate better solutions than a short-term only tabu search method because of the longer-term memory based tabu mechanism.

Starting from the same initial solution, Experiment 5 can generate better solutions than Experiment 2, which means the transition frequency based longer-term memory, together with other techniques such as tabu relaxation and move preference discussed in Chapter 5, can work well. We also noticed that the improvement is between 3% to 20%. Because of the differences in the density of matrix, the three instances behave quite differently with the OTTABU algorithm. The data quoted are from Table 8.3.1, 8.3.4, 8.3.5, 8.3.7, 8.4.1, 8.4.4, 8.4.5, 8.4.7, 8.5.1, 8.5.4, 8.5.5, and 8.5.7. The comparisons are presented in the table 8.2.2 and Table 8.2.3.

Starting from s_{bp0}	Experiment 2	Experiment 5	Improvement
UTA-S-92	0, 3552	0, 3347	6%
CAR-F-92	0, 2798	0, 2549	9%
OTT-F-96	0, 693	0, 555	20%

Table 8.2.2 TS only vs OTTABU (Starting from s_{bp0})

Starting from $s_{restruc0}$	Experiment 2	Experiment 5	Improvement
UTA-S-92	0, 2932	0, 2811	4%
CAR-F-92	0, 2090	0, 2022	3%
OTT-F-96	0, 631	0, 512	19%

Table 8.2.3 TS only vs OTTABU (Starting from $s_{restruc0}$)

3. OTTASYS can generate much better solutions than a one-phase and recency-based tabu method.

In the instance of UTA-S-92, the best solution obtained by *TS* only and a single phase is (0, 3645). We choose it from the two solutions, one is (0, 3683) with neighborhood *N*(s)* in Table 8.3.1, another one is (0, 3645) with neighborhood *N(s)* in Table 8.3.2. If both *TS* and *TL* are used, and we start from a restructured initial solution, the best solution obtained is (0, 2769) in Table 8.3.7. Comparing the two solutions, we found OTTASYS can improve the solution better than the *TS* only algorithm. The solution is improved by 25%.

In the instance of CAR-F-92, the best solutions obtained by *TS* only and a single phase is (0, 2830). We choose it from the two solutions, one is (0, 2883) with

neighborhood *N*(s)* in Table 8.4.1, another one is (0, 2830) with neighborhood *N(s)* in Table 8.4.2. If both *TS* and *TL* are used, and we start from a restructured initial solution, the best solution obtained is (0, 1922) in Table 8.4.7. Comparing the two solutions, we found OTTASYS can improve the solution better than the *TS* only algorithm. The solution is improved by 32%.

In the instance of OTT-F-96, the best solution obtained by *TS* only and a single phase is (0, 789). We choose it from the two solutions. One is (0, 789) with neighborhood *N*(s)* and another one has a same value (0, 789) with neighborhood *N(s)*. If both *TS* and *TL* are used, and we start from a restructured initial solution, the best solution obtained is (0, 512). Comparing the two solutions, we found OTTASYS can obtain a better solution than a *TS* only algorithm. The solution is improved by 35%. The data quoted here can be found in Table 8.5.1, 8.5.2, and 8.5.7.

4. OTTABU needs to be improved

When we continued the optimization from the solutions in Result 5 and Result 9, we found that we can not get better average solutions than we have. Even starting from the solutions in Result 5, we can only get a few better solutions, but these new solutions still are much worse than the solutions in Result 9. This experiment shows OTTABU alone can not move away from local optima it has reached to explore new search space more aggressively.

§8.3 Experiments and Results: UTA-S-92

General parameters

By analyzing this examination set, we have found there are about 308 smallest examinations whose accumulative percentage is 15%. So, the initial value of l_{TL} is set to 308, and the *nbmax* is set to 924.

Initial Solution

Result 1 (Experiment 1): The solution obtained by the bin-packing algorithm from the raw data is s_{bp}. The solution s_{bp} (0, 5998) is a feasible solution. In other words, s_{bp0} is s_{bp}. There are 5998 second-order conflicts found in this solution. So, starting from this solution, we move directly to other experiments.

Optimization of s_{bp0} (0, 5998)

Result 2: generated by Experiment 2 from initial solution s_{bp0}, and presented in Table 8.3.1.

Solution	S_{3500}	S_{3552}	S_{3560}	S_{3564}	S_{3586}	Average
Pass 1 (NS*)	0, 3697	0, 3700	0, 3683	0, 3684	0, 3698	**0, 3692**
Pass 2 (NS)	0, 3500	0, 3552	0, 3560	0, 3564	0, 3586	**0, 3552**

Table 8.3.1 Result 2 of UTA-S-92

Result 3: generated by Experiment 3 from initial solution s_{bp0}, and presented in Table 8.3.2

Solution	S_{3618}	S_{3620}	S_{3620b}	S_{3621}	S_{3629}	Average
Pass 1 (NS)	0, 3645	0, 3645	0, 3645	0, 3645	0, 3650	**0, 3646**
Pass 2 (NS*)	0, 3618	0, 3620	0, 3620	0, 3621	0, 3629	**0, 3622**

Table 8.3.2 Result 3 of UTA-S-92

Result 4: generated by Experiment 4 from initial solution s_{bp0}, and presented in Table 8.3.3

Solution	S_{3325}	S_{3376}	S_{3420}	S_{3426}	S_{3467}	Average
Pass 1 (NS*)	0, 3775	0, 3750	0, 3749	0, 3775	0, 3754	**0, 3761**
Pass 2 (NS)	0, 3325	0, 3376	0, 3420	0, 3426	0, 3467	**0, 3409**

Table 8.3.3 Result 4 of UTA-S-92

Result 5: generated by Experiment 5 from initial solution s_{bp0}, and presented in Table 8.3.4

Solution	S_{3294}	S_{3313}	S_{3358}	S_{3378}	S_{3395}	Average
Pass 1 (NS*)	0, 3518	0, 3465	0, 3566	0, 3582	0, 3555	**0, 3537**
Pass 2 (NS)	0, 3294	0, 3313	0, 3358	0, 3378	0, 3395	**0, 3347**

Table 8.3.4 Result 5 of UTA-S-92

Restructure of s_{bp0} (0, 8668)

Result 6 (Experiment 6): Restructuring s_{bp0} to get an infeasible solution s_{inf} (225, 1159), then apply Experiment 1 to obtain a feasible solution $s_{restruc}$ (0, 4179) from this infeasible solution.

Optimization of $s_{restruc}$ (0, 4179)

Result 7: generated by Experiment 2 from an initial solution $s_{restruc0}$ and presented in Table 8.3.5

Solution	S_{2920}	S_{2928}	S_{2933}	S_{2937}	S_{2943}	Average
Pass 1 (NS*)	0, 3015	0, 3018	0, 3036	0, 3018	0, 3025	**0, 3022**
Pass 2 (NS)	0, 2920	0, 2928	0, 2933	0, 2937	0, 2943	**0, 2932**

Table 8.3.5 Result 7 of UTA-S-92

Result 8: generated by Experiment 4 from an initial solution $s_{restruc0}$ and presented in Table 8.3.6

Solution	S_{2985}	S_{3001}	S_{3001b}	S_{3030}	S_{3030b}	Average
Pass 1 (NS*)	0, 2985	0, 3001	0, 3001	0, 3030	0, 3030	**0, 3009**
Pass 2 (NS)	0, 2985	0, 3001	0, 3001	0, 3030	0, 3030	**0, 3009**

Table 8.3.6 Result 8 of UTA-S-92

Result 9: generated by Experiment 5 from an initial solution $s_{restruc0}$ and presented in Table 8.3.7.

Solution	S_{2769}	S_{2799}	S_{2816}	S_{2833}	S_{2839}	Average
Pass 1 (NS*)	0, 2936	0, 2985	0, 2989	0, 2996	0, 2989	**0, 2979**
Pass 2 (NS)	0, 2769	0, 2799	0, 2816	0, 2833	0, 3839	**0, 2811**

Table 8.3.7 Result 9 of UTA-S-92

Result 10: generated by Experiment 7 from the solution $s_{restruc0}$ and presented in Table 8.3.8

Solution	S_{2747}	S_{2753}	S_{2817}	S_{2835}	S_{2857}	Average
Pass 1 (NS*)	0, 2975	0, 2951	0, 2985	0, 3030	0, 3001	**0, 2988**
Pass 2 (NS)	0, 2747	0, 2753	0, 2817	0, 2835	0, 2857	**0, 2802**

Table 8.3.8 Result 10 of UTA-S-92

Result 11 generated by Experiment 8 from the solution s_{bp0} and presented in Table 8.3.9

Solution	S_{3455}	S_{3509}	S_{3551}	S_{3554}	S_{3590}	Average
Pass 1 (NS*)	0, 3698	0, 3697	0, 3685	0, 3686	0, 3713	**0, 3696**
Pass 2 (NS)	0, 3455	0, 3509	0, 3551	0, 3554	0, 3590	**0, 3532**

Table 8.3.9 Result 10 of UTA-S-92

More results: apply Experiment 4 to solution s_{2816} to generate more solutions, some of them are very good, in fact, s_{2700} is the best solution obtained so far. They are listed in Table 8.3.10.

Solution	S_{2700}	S_{2765}	S_{2811}	S_{2816b}	-	Average
Pass 1 (NS*)	0, 2816	0, 2816	0, 2816	0, 2816	-	
Pass 2 (NS)	0, 2700	0, 2765	0, 2811	0, 2816	-	-

Table 8.3.10 Some Result generated from an optimal solution

§8.4 Experiments and Results: CAR-F-92

General parameters

By analyzing this examination set, we found there are about 253 smallest examinations which accumulative percentage is 15%. So, the initial value of l_{TL} is set to 253, and the *nbmax* is set to 759.

Initial Solution

Result 1 (Experiment 1): The solution obtained by the bin-packing algorithm from the raw data is s_{bp}. The solution s_{bp} (0, 8668) is a feasible solution. In other words, s_{bp0} is s_{bp}. There are 8668 second-order conflicts found in this solution. So, starting from this solution, we move directly to other experiments.

Optimization of s_{bp0} (0,8668)

Result 2: generated by Experiment 2 from an initial solution s_{bp0} and presented in Table 8.4.1

Solution	S_{2758}	S_{2776}	S_{2802}	S_{2820}	S_{2833}	Average
Pass 1 (NS*)	0, 2923	0, 2883	0, 2921	0, 2904	0, 2906	**0, 2907**
Pass 2 (NS)	0, 2758	0, 2776	0, 2802	0, 2820	0, 2833	**0, 2798**

Table 8.4.1 Result 2 of CAR-F-92

Result 3: generated by Experiment 3 from an initial solution s_{bp0} and presented in Table 8.4.2

Solution	S_{2780}	S_{2786}	S_{2805}	S_{2819}	S_{2833}	Average
Pass 1 (NS)	0, 2830	0, 2855	0, 2855			
Pass 2 (NS*)	0, 2780	0, 2786	0, 2805	0, 2819	0, 2833	**0, 2814**

Table 8.4.2 Result 3 of CAR-F-92

Result 4: generated by Experiment 4 from an initial solution s_{bp0} and presented in Table 8.4.3

Solution	S_{2539}	S_{2617}	S_{2625}	S_{2629}	S_{2731}	Average
Pass 1 (NS*)	0, 2914	0, 2914	0, 2914	0, 2914	0, 2914	**0, 2914**
Pass 2 (NS)	0, 2539	0, 2617	0, 2625	0, 2629	0, 2731	**0, 2628**

Table 8.4.3 Result 4 of CAR-F-92

Result 5: generated by Experiment 5 from an initial solution s_{bp0} and presented in Table 8.4.4

Solution	S_{2490}	S_{2504}	S_{2541}	S_{2569}	S_{2639}	Average
Pass 1 (NS*)	0, 2644	0, 2677	0, 2697	0, 2691	0, 2806	**0, 2703**
Pass 2 (NS)	0, 2490	0, 2504	0, 2541	0, 2569	0, 2639	**0, 2549**

Table 8.4.4 Result 5 of CAR-F-92

Restructure of s_{bp0} (0, 8668)

Result 6 (Experiment 6): Restructuring s_{bp0} to get an infeasible solution s_{inf} (543, 160), then apply Experiment 1 to obtain a feasible solution $s_{restruc}$ (0, 6668) from this infeasible solution.

Optimization of $s_{restruc}$ (0, 6668)

Result 7: generated by Experiment 2 from an initial solution $s_{restruc0}$ and presented in Table 8.4.5

Solution	S_{2079}	S_{2084}	S_{2087}	S_{2092}	S_{2118}	Average
Pass 1 (NS*)	-	0, 2205	0, 2203	0, 2218	0, 2211	**0, 2209**
Pass 2 (NS)	0, 2070	0, 2084	0, 2087	0, 2092	0, 2118	**0, 2090**

Table 8.4.5 Result 7 of CAR-F-92

Result 8: generated by Experiment 4 from an initial solution $s_{restruc0}$ and presented in Table 8.4.6

Solution	S_{1986}	S_{2016}	S_{2071}	S_{2090}	S_{2103}	Average
Pass 1 (NS*)	0, 2220	0, 2220	0, 2220	0, 2220	0, 2220	**0, 2220**
Pass 2 (NS)	0, 1986	0, 2016	0, 2071	0, 2090	0, 2103	**0, 2053**

Table 8.4.6 Result 8 of CAR-F-92

Result 9: generated by Experiment 5 from an initial solution $s_{restruc0}$ and presented in Table 8.4.7

Solution	S_{1922}	S_{1989}	S_{2064}	S_{2064b}	S_{2072}	Average
Pass 1 (NS*)	0, 2220	0, 2220	0, 2220	0, 2220	0, 2220	**0, 2220**
Pass 2 (NS)	0, 1922	0, 1989	0, 2064	0, 2064	0, 2072	**0, 2022**

Table 8.4.7 Result 9 of CAR-F-92

Other solutions

Result 10: generated by Experiment 7 from the solution $s_{restruc0}$ and presented in Table 8.4.8

Solution	S_{1932}	S_{1997}	S_{2039}	S_{2050}	S_{2075}	Average
Pass 1 (NS*)	0, 2220	0, 2220	0, 2220	0, 2220	0, 2220	**0, 2220**
Pass 2 (NS)	0, 1932	0, 1997	0, 2039	0, 2050	0, 2075	**0, 2018**

Table 8.4.8 Result 10 of CAR-F-92

Result 11: generated by Experiment 7 from the solution s_{bp0} and presented in Table 8.4.9

Solution	S_{2732}	S_{2761}	S_{2815}	S_{2816}	S_{2817}	Average
Pass 1 (NS*)	0, 2824	0, 2882	0, 2914	0, 2924	0, 2913	**0, 2891**
Pass 2 (NS)	0, 2732	0, 2761	0, 2815	0, 2816	0, 2817	**0, 2788**

Table 8.4.9 Result 11 of CAR-F-92

§8.5 Experiments and Results: OTT-F-96

General parameters

By analyzing this examination set, we found there are about 354 smallest examinations whose accumulative percentage is 15%. So, the initial value of l_{TL} is set to 354, and the *nbmax* is set to 1062.

Initial Solution s_{bp0} (0, 8686)

Result 1 (Experiment 1): The solution obtained by the bin-packing algorithm from the raw data is s_{bp}. The solution s_{bp} (0, 8686) is a feasible solution. In other words, s_{bp0} is s_{bp}. There are 8686 second-order conflicts found in this solution. So, starting from this solution, we move directly to other experiments.

Result 2: generated by Experiment 2 from an initial solution s_{bp0} and listed in Table 8.5.1.

Solution	S_{671}	S_{692}	S_{693}	S_{697}	S_{711}	Average
Pass 1 (NS*)	0, 800	0, 801	0, 807	0, 807	0, 780	**0, 789**
Pass 2 (NS)	0, 671	0, 692	0, 693	0, 697	0, 711	**0, 693**

Table 8.5.1. Result 2 of OTT-F-96

Result 3: generated by Experiment 3 from an initial solution s_{bp0} and listed in Table 8.5.2

Solution	S_{714}	S_{716}	S_{724}	S_{736}	S_{741}	Average
Pass 1 (NS)	0, 782	0, 782	0, 782	0, 782	0, 782	**0, 782**
Pass 2 (NS*)	0, 714	0, 716	0, 724	0, 736	0, 741	**0, 726**

Table 8.5.2. Result 3 of OTT-F-96

Result 4: generated by Experiment 4 from an initial solution s_{bp0} and listed in Table 8.5.3

Solution	S_{507}	S_{522}	S_{548}	S_{550}	S_{555b}	Average
Pass 1 (NS*)	0, 827	0, 827	0, 827	0, 827	0, 827	**0, 827**
Pass 2 (NS)	0, 507	0, 522	0, 548	0, 550	0, 555	**0, 549**

Table 8.5.3. Result 4 of OTT-F-96

Result 5: generated by Experiment 5 from an initial solution s_{bp0} and listed in Table 8.5.4.

Solution	S_{542b}	S_{551}	S_{553}	S_{560}	S_{573}	Average
Pass 1 (NS*)	0, 721	0, 730	0, 733	0, 730	0, 768	**0, 736**
Pass 2 (NS)	0, 542	0, 551	0, 553	0, 560	0, 573	**0, 555**

Table 8.5.4. Result 5 of OTT-F-96

Restructure of s_{bp0} (0, 8668)

Result 6 (Experiment 6): Restructuring s_{bp0} to get an infeasible solution s_{inf} (65, 139), then apply Experiment 1 to obtain a feasible solution $s_{restruc0}$ (0, 1173) from this infeasible solution.

Optimization of $s_{restruc0}$ (0, 1173)

Result 7: generated by Experiment 2 from an initial solution $s_{restruc0}$ and listed in table 8.5.5.

Solution	S_{605}	S_{615}	S_{627}	S_{643}	S_{666}	Average
Pass 1 (NS*)	0, 683	0, 696	0, 693	0, 707	0, 738	**0, 703**
Pass 2 (NS)	0, 605	0, 615	0, 627	0, 643	0, 666	**0, 631**

Table 8.5.5. Result 7 of OTT-F-96

Result 8: generated by Experiment 4 from an initial solution $s_{restruc0}$ and listed in table 8.5.6.

Solution	S_{493}	S_{526}	S_{542}	S_{555}	S_{559}	Average
Pass 1 (NS*)	0, 750	0, 702	0, 702	0, 702	0, 702	**0, 712**
Pass 2 (NS)	0, 493	0, 526	0, 542	0, 555	0, 559	**0, 535**

Table 8.5.6. Result 8 of OTT-F-96

Result 9: generated by Experiment 5 from an initial solution $s_{restruc0}$ and listed in table 8.5.7

Solution	S_{469}	S_{494}	S_{520}	S_{528}	S_{548}	Average
Pass 1 (NS*)	0, 639	0, 671	0, 676	0, 678	0, 704	**0, 674**
Pass 2 (NS)	0, 469	0, 494	0, 520	0, 528	0, 548	**0, 512**

Table 8.5.7 Result 9 of OTT-F-96

Other solutions

Result 10: generated by Experiment 7 from $s_{restruc0}$ (with threshold r is set to 15%) and listed in Table 8.5.8.

Solution	S_{489}	S_{496}	S_{516}	S_{528}	S_{533}	Average
Pass 1 (NS*)	0, 665	0, 702	0, 702	0, 702	0, 702	**0, 698**
Pass 2 (NS)	0, 489	0, 496	0, 516	0, 528	0, 533	**0, 512**

Table 8.5.8 Result 10 of OTT-F-96

Result 11: generated by Experiment 8 from s_{bp0} (tabu relaxation enabled) and listed in Table 8.5.9.

Solution	S_{2732}	S_{2761}	S_{2815}	S_{2816}	S_{2817}	Average
Pass 1 (NS*)	0, 756	0, 797	0, 818	0, 807	0, 769	**0, 789**
Pass 2 (NS)	0, 666	0, 681	0, 687	0, 699	0, 703	**0, 687**

Table 8.5.9 Result 11 of OTT-F-96

Execution time

Running on a Pentium III 533 IBM PC, OTTABU takes at most 9 minutes to complete a UTA-S-92 solution shown in the experiments above, 7 minutes for a CAR-F-92 solution, or 15 minutes for a OTT-F-96 solution.

Chapter 9

Applications and Comparisons

Two applications of OTTASYS are presented in this chapter. We apply the procedure developed in the previous chapters to solve a problem to find a conflict-free solution while spreading student exams as evenly as possible over the schedule. In paper [CarterLL96], Carter *et al.* defined this problem and presented several algorithm strategies and their test results based on a dozen of sets of real data. Recently, Di Gaspero and Schaerf have developed a tabu search strategy to schedule the real data and have published the results as well [GasperoS00]. Our experiments for UTA-S92 and CAR-F-92 (arbitrarily chosen two sets of data within the publicly available data) have shown that OTTASYS can generate very high quality solutions. Our solution to CAR-F-92 is much better than Di Gaspero and Schaerf's.

§9.1 Problem Description

For a given set of exams to be scheduled and the number of time slots, we construct a conflict-free examination timetable (*i.e.*, no student is asked to take two exams at a time) while spreading student exams as evenly as possible over the schedule. In order to avoid too many secondary parameters affecting our results and conclusions, we do not consider any side constraints. In other words, the problem is modeled as an unconstrained problem. For example, there are no constraints for the maximum seats, no constraints for the maximum exams per timeslot, and no pre-assigned exams.

The objective function, or the cost function, given by Carter, Laporte and Lee [CarterLL96], is expressed as:

$$f(x) = 16W_1 + 8W_2 + 4W_3 + 2W_4 + W_5 \qquad (9.1)$$

where W_i (i=1, 2, 3, 4, and 5) is defined as the number of students who have to take two exams, i timeslots apart. For instance, W_1 is the number of students taking two consecutive exams.

In this model, the first order conflicts are modeled as hard constraints. The second order conflicts are treated as soft constraints, and included into the objective function f that measures the quality of the solution. A measurement factor called *cost* is introduced to measure and assess the quality of potential schedules. Assuming that consecutive time slots lie one unit apart, we define f as assigning a proximity cost w_i whenever a student has to take two exams separated by i time slots. The cost of each conflict is then multiplied by the number of students involved in both exams. The cost decreases from 16 to 1 as follows: $w_1=16$, $w_2=8$, $w_3=4$, $w_4=2$, and $w_5=1$. The *cost* function is then normalized based on the total number of students, *i.e.*, the average cost per student is used to evaluate the quality of a solution instead of the cost function value.

§9.2 Test Data, Methods and Results: A Summary

We conduct the tests with two instances of Carter's public data, UTA-S-92 and CAR-F-92. For UTA-S-92, we must assign 622 exams into 35 time slots, and for CAR-F-92, we need to schedule 543 exams into 32 time slots.

Carter *et al* and their EXAMINATION

With EXAMINE, a platform built and used by Carter *et al* [CarterLC94, CarterLL96], Carter used forty different strategies to test some real test data. The strategies consist of five basic sorting rules, using cliques or not, using costs or

not, using backtracking or not. The five basic sorting rules are the largest degree (LD), the saturation degree (SD), the largest weighted degree (LWD), the largest enrollment (LE), and the random ordering (RO). The details can be found in [CarterLL96].

Di Gaspero and Schaerf and their on-going TS algorithm

Di Gaspero and Schaerf did a study with tabu search techniques (short term memory only) with Carter's test data [GasperoS00]. Their best results have been obtained by setting the tabu tenure varying randomly between 15 and 25. The stop criterion is based on the number of iterations from the last improvement (idle iterations). The number of idle iterations depends on the instance, varying from 2000 to 20000.

OTTASYS

We have used the multi-phase procedure introduced in Chapter 7 to test the two instances. For each set of test data, an initial solution is generated by the bin-packing algorithm in Phase 1. Because there are no constraints on the maximum number of seats for each time slot, we get a feasible solution very easily. We restructure the solution obtained in Phase One with small penalty for the first order conflicts into an infeasible one. In order to get a restructured feasible solution, we apply OTTABU again to this infeasible solution. This work was done in Phase three. In Phase Four, we built an objective function according to formula (f.9.1), and apply OTTABU to generate an optimal solution. We run Phase Four many times and get a list of solutions. Details can be found in section §9.4 and §9.5.

Table 9.1 summarizes the performances of our test result of OTTASYS with respect to Carter's benchmarks and Di Gaspero and Schaerf's experiments.

Timetable	*Timeslots*	*OTTASYS Results*		*Di Gaspero and Schaerf's TS Results*		*Carter's Results*
		Average cost	*Best cost*	*Average cost*	*Best cost*	*Average Costs for the different strategies*
CAR-F-92	32	4.66	4.53	5.6	5.2	6.2 - 7.6
UTA-S-92	35	3.97	3.90	4.5	4.2	3.5 - 4.5

Table 9.1 Comparison with other researchers' results

§9.3 Discussions

From the result listed above, we can find that:

1) OTTASYS can generate much better solutions than Di Gaspero and Schaerf's results as shown in Table 9.2.

Solutions	OTTASYS	Gaspero's TS only	Improvement
CAR-F-92	4.66	5.6	17%
UTA-S-92	3.97	4.5	12%

Table 9.2. OTTASYS can improve TS only method

2. We can conclude that even though OTTASYS has a different performance for different test data, it can generate good quality solutions better than the average, for instance, in UTA-S-92.

3. It has been proved, in this application, that OTTASYS can be used in its general form to generate solutions with high quality. We do not need to adjust the parameters to adapt to the different examination instances; just follow the simple

procedure: generate an initial solution, and restructure it, then apply OTTABU to optimize it.

4. We noticed that very big *nbmax*'s (from 2,000 to 20,000 iterations) are used in the Di Gaspero's algorithm [GasperoS00]. Our experiments have shown that a very large *nbmax* does nothing to improve TS only Tabu Search but wastes search iterations due to cycling.

In addition, a four-pass OTTABU was used for this experiment rather than a two-pass OTTABU. The former took more time to run.

5. Because there are no constraints on the maximum number of seats per timeslot, the heavy exams (*i.e.*, these nodes with large enrollments, degrees, and/or weights) would be moved more often than those in the experiments presented in Chapter 8, if such a move can generate a better solution. We found the results listed in Table 9.5 and Table 9.9 are much better than the results listed in Table 8.3.7 and Table 8.4.7 respectively in Chapter 8, which may show that moving heavier exams frequently might increase the likelihood of producing better solutions.

6. Compared with the short-term memory methods, the longer-term memory can help to generate better solutions as shown by the tables in the section 9.4 and 9.5.

§9.4 Test Data and Results: UTA-S-92

There are 622 examinations taken by 21266 students. The total enrollment is 58979, the total number of edges is 24249; each student takes 2.77 exams on average. The matrix density is 0.126.

Constraints and Parameters for OTTABU

The maximum number of timeslots is 35; there are no constraints for either maximum seats or maximum rooms per timeslot. Neighborhood types are chosen

in this way: *N*(s)* in Pass 1, *N(s)* in Pass 2, *N*(s)* in Pass 3, and *N(s)* in Pass 4. Random tenures ranging from 15 to 25 are used for the short term tabu list (*TS*). Cumulative percentage is set to 15% to estimate the most active or the lightest exams, which means the size of longer term tabu list (*TL*) is set to 308. When both *TL* and *TS* are activated, the *nbmax* in both Pass 1 and Pass 3 is set to 924, but it is 143 in both Pass 2 and Pass 4. Tabu relaxation is activated when either the number of null iterations is equal to one-third of *nbmax* or the uphill movement reaches the threshold (it is set to 7%).

Feasible initial solution

Get an infeasible solution S_{bp} with a bin-packing algorithm, then apply OTTABU to S_{bp} to get a feasible solution S_{00} with the following parameters: penalty w_0 is set to 1, w_1, w_2, w_3, w_4, and w_5 are set to 0s. *TL* is disabled, only *TS* is used and *nbmax* is 270. The results are listed in Table 9.3.

Solution	*From*	*W0*	*W1*	*W2*	*W3*	*W4*	*W5*	*f(s)*	*Cost*
S_{bp}	raw data	23	6166	6866	5539	5307	5261		
S_{00}	S_{bp}	0	5799	6270	5122	4935	4926	178228	8.38

Table 9.3. Feasible Initial Solution

Restructuring the initial solution

Starting from S_{00}, apply OTTABU to reconstruct the feasible initial solution. An infeasible solution S_{87124i} is obtained. The penalty w_0 is set to 128, w_1, w_2, w_3, w_4, and w_5 are set to 16, 8, 4, 2, and 1, respectively. *TL* is disabled, only *TS* is used and *nbmax* is 270.

Then apply OTTABU to get a feasible solution S_{0w} from S_{87124i}. The following parameters are set in this round: penalty w_0 is set to 1, w_1, w_2, w_3, w_4, and w_5 are

set to 0s. *TL* is disabled, only *TS* is used and *nbmax* is 270. The results are listed in Table 9.4.

Solution	*From*	*W0*	*W1*	*W2*	*W3*	*W4*	*W5*	*f(s)*	*Cost*
S_{87124i}	S_{00}	31	2267	2773	3476	3645	3506	87124	
S_{0w}	S_{87124i}	0	3055	3231	3605	3337	3961	99783	4.69

Table 9.4. Restructure of initial solution

Both longer term and short term memory used

Starting from S_{0w}, apply OTTABU to optimize this feasible initial solution, five solutions S_{87047} and S_{87047b} are obtained. The penalty w_0 is set to 20000, w_1, w_2, w_3, w_4, and w_5 are set to 16, 8, 4, 2, and 1, respectively. Both *TL* and *TS* are used. The tenures of *TS* are set to 15-25 randomly and the size of *TL* is 308 (i.e., the cumulative percentage is 15%). The average cost is 3.97 and the best is 3.93. The optimal solutions are given in Table 9.5.

Solution	*From*	*W0*	*W1*	*W2*	*W3*	*W4*	*W5*	*f(s)*	*Cost*
S_{83654}	S_{0w}	0	2279	2727	3693	3447	3708	83654	3.93
S_{83982}	S_{0w}	0	2222	2895	3561	3572	3882	83982	3.95
S_{84469}	S_{0w}	0	2323	2724	3646	3491	3943	84469	3.97
S_{84544}	S_{0w}	0	2335	2798	3576	3322	3852	84544	3.98
S_{85173}	S_{0w}	0	2367	2799	3575	3326	3957	85173	4.00

Table 9.5. Optimal Solutions of UTA-S-92

In search of better solutions

Applying the same procedure above, several better solutions were obtained, from S_{83654} to S_{83136}, from S_{83136} to S_{82998}. These solutions are not used for comparison in this paper, so they are not listed.

Comparison with short-term memory tabu list only

In order to do further comparison of the *TS* only tabu search and both *TS* and *TL* tabu search, we disable *TL* and tabu relaxation, then run five solutions. The procedure is following:

Starting from S_{0w}, apply OTTABU to optimize this feasible initial solution, two solutions S_{87047} and S_{87047b} are obtained. The penalty w_0 is set to 20000, w_1, w_2, w_3, w_4, and w_5 are set to 16, 8, 4, 2, and 1 respectively. *TL* is disabled, only *TS* is used and *nbmax* is 270. Apply the same procedure except the *nbmax* is set to 924, a solution S_{87045} is obtained from S_{0w}. The average cost is 4.09 and the best is 4.09. The results are given Table 9.6.

Solution	*From*	*W0*	*W1*	*W2*	*W3*	*W4*	*W5*	*f(s)*	*Cost*
S_{87047}	S_{0w}	0	2439	2853	3651	3395	3805	87047	4.09
S_{87047b}	S_{0w}	0	2440	2849	3659	3386	3807	87047	4.09
S_{87045}	S_{0w}	0	2440	2848	3658	3395	3799	87045	4.09

Table 9.6. A comparison of UTA-S-92

Compared with the TS only algorithm, the longer-term memory based tabu search can improve the solution by 3%.

§9.5 Test Data and Results: CAR-F-92

There are 543 examinations taken by 18419 students. The total enrollment is 55522, the total number of edges is 20305, each student takes 3.01 exams on average. The matrix density is 0.138.

Constraints and parameters for OTTABU

The maximum number of timeslots is 32, there are no constraints for either maximum seats or maximum classrooms (i.e., exams) per timeslot; Neighborhood types are chosen in this way: $N^*(s)$ in Pass 1, $N(s)$ in Pass 2, $N^*(s)$ in Pass 3, and $N(s)$ in Pass 4. Random tenures ranging from 15 to 25 are set to the short term tabu list (*TS*). Cumulative percentage is set to 15% to estimate the most active or the lightest exams, which means the size of longer term tabu list (*TL*) is set to 253. When both *TL* and *TS* are activated, the *nbmax* in both Pass 1 and Pass 3 is set to 759, but it is 126 in both Pass 2 and Pass 4. Tabu relaxation is activated when either the number of null iteration is equal to one-third of *nbmax* or the uphill movement reaches the threshold (it is set to 7%). For some tests, the threshold may be disabled.

Feasible initial solution

Get an infeasible solution S_{bp} with a bin-packing algorithm, then apply OTTABU to S_{bp} to get an infeasible solution S_{0i} with following parameters: penalty w_0, w_1, w_2, w_3, w_4, and w_5 are set to 32, 16, 8, 4, 2, and 1, respectively. Both *TL* and *TS* are used. Then apply the same procedure to this infeasible solution (with penalties: w_0=1 and w_1=w_2=w_3=w_4=w_5=0), a feasible solution S_{0c} is obtained. Because S_{0i} is obtained with a very small penalty for w_0, there is no need for restructuring solution S_{0c}.

Solution	*From*	*W0*	*W1*	*W2*	*W3*	*W4*	*W5*	*f(s)*	*Cost*
S_{bp}	raw data	110	7173	5547	5639	5836	5128		
S_{0i}	S_{bp}	186	1327	2144	4030	3872	3242		
S_{0c}	S_{0i}	0	4911	4102	4800	4190	3792	142764	7.75

Table 9.7. Feasible Initial Solution

Both longer-term and short-term memory used

Starting from S_{0w}, applying OTTABU to optimize this feasible initial solution, five solutions are obtained. The penalty w_0 is set to 20000, w_1, w_2, w_3, w_4, and w_5 are set to 16, 8, 4, 2, and 1, respectively. Both *TL* and *TS* are used. The tenures of *TS* are set to 15-25 randomly and the size of *TL* is 253 (*i.e.*, the cumulative percentage is 15%). The threshold is disabled. The average cost is 4.69 and the best cost is 4.59.The optimal solutions are listed in Table 9.9.

Solution	*From*	*W0*	*W1*	*W2*	*W3*	*W4*	*W5*	*f(s)*	*Cost*
S_{84598}	S_{0c}	0	2214	2845	3706	3691	4208	84598	4.59
S_{85534}	S_{0c}	0	2229	2818	3916	3759	4144	85534	4.64
S_{86443}	S_{0c}	0	2313	2842	3830	3600	4179	86443	4.69
S_{87566}	S_{0c}	0	2331	3012	3718	3775	3752	87566	4.75
S_{87734}	S_{0c}	0	2357	2912	3785	3853	3889	87743	4.76

Table 9.8. Restructure of initial solution

With enabling the threshold and repeating the same procedure, another six solutions are obtained. The average cost is 4.66 and the best is 4.59. The optimal solutions are given in Table 9.9.

In search of better solutions

Applying the same procedure above, better solutions are obtained, from S_{84577} to S_{83628}, from S_{83628} to S_{83527}. The cost of S_{83527} is 4.53.

Comparison with Short term tabu list only

In order to do further comparison of the TS only tabu search and both TS and TL tabu search, we disable TL and tabu relaxation, then generate five solutions. The procedure is the following:

Starting from S_{0w}, apply OTTABU to optimize this feasible initial solution; five solutions S_{94502}, S_{94502b}, S_{94503}, S_{94503b} and S_{94504} are obtained. The penalty w_0 is set to 20000, w_1, w_2, w_3, w_4, and w_5 are set to 16, 8, 4, 2, and 1 respectively. *TL* is disabled, only *TS* is used and *nbmax* is 759. The threshold of controlling uphill movement is disabled. The average cost is 5.13, the best is 5.13. Compared with the *TS* only algorithm, the longer term tabu search can improve the solution by 9%

Solution	*From*	*W0*	*W1*	*W2*	*W3*	*W4*	*W5*	*f(s)*	*Cost*
S_{84577}	S_{0c}	0	2238	2742	3870	3744	3865	84577	4.59
S_{84670}	S_{0c}	0	2180	2861	3776	3945	3908	84670	4.60
S_{85240}	S_{0c}	0	2252	2795	3872	3475	4410	85240	4.63
S_{85349}	S_{0c}	0	2208	2884	3904	3646	4041	85349	4.63
S_{87249}	S_{0c}	0	2277	2966	4014	3542	3949	87249	4.74
S_{87452}	S_{0c}	0	2315	3006	3798	3586	4000	87452	4.75

Table 9.9. Optimal Solutions of CAR-F-92

Solution	*From*	*W0*	*W1*	*W2*	*W3*	*W4*	*W5*	*f(s)*	*Cost*
S_{94502}	S_{0c}	0	2579	3178	4124	3706	3906	94502	5.13
S_{94502b}	S_{0c}	0	2579	3178	4125	3705	3904	94502	5.13
S_{94503}	S_{0c}	0	2579	3175	4130	3690	3939	94503	5.13
S_{94503b}	S_{0c}	0	2579	3178	4127	3701	3905	94503	5.13
S_{94504}	S_{0c}	0	2582	3175	4122	3701	3902	94504	5.13

Table 9.10. A comparison of CAR-F-92

Execution time

Running on a Pentium III 533 IBM PC, starting from the raw data and going through the four phases, OTTASYS takes approx. two hours to generate an optimal UTA-S-92 solution listed in Table 9.5. It takes less than one minute in Phase 1 (no Phase 2 is needed because a feasible solution has been generated), about 60 minutes in Phase 3, and 50 minutes in Phase 4.

Since CAR-F-92 has less data and a smaller matrix density than UTA-S-92, OTTASYS generates its best CAR-F-92 solution listed in Table 9.9 in one and half hours.

Chapter 10

Conclusions and Future Work

Based on our experimental tests using real examination data sets and the comparison of them with the results presented by other researchers, we conclude that:

The OTTASYS can generate higher quality solutions compared with results generated by the recency-based tabu search techniques presented by Di Gaspero and Schaerf [GasperoS00] and the ones reported by Carter *et al.* [CarterLL96] with the backtracking techniques as discussed in Chapter 9.

In the OTTABU algorithm, we use two tabu lists. One is TS which is a recency-based short-term memory structure that is commonly used and functions well in other applications; another one is TL which is built by a move (or transition) frequency-based longer term memory and is used to prevent cycling and diversify the search space effectively to help get better solutions. In fact, the combination of the two tabu lists always generates better solutions than the short term memory tabu list only.

The effects of the relaxation of the tabu lists were found to be quite pronounced. In the OTTABU algorithm, if a given number of iterations has elapsed and the longer term tabu list 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 tabu lists. Relaxation of the tabu lists will change the neighborhood of the current solution dramatically, which may drive the search into a new region and increase the likelihood of finding a better solution.

The solution restructuring phase, integrated in the OTTASYS, is very important and productive because it can improve the solution dramatically even though it is still not clear why and how it improves solutions. We think it functions like

another Tabu Search technique called strategic oscillation. According to some successful applications of Tabu Search techniques in some other optimization problems, better solutions could be generated by driving solutions to cross feasible-infeasible borders back and forth. The term strategic oscillation refers to such a process.

In the experiments presented in Chapter 8 and 9, we give several examples which show how some not so good solutions are optimized to high quality solutions by applying solution restructuring technique with OTTABU algorithm.

The OTTASYS works as a systematic approach which can be automated. This system can generate an initial solution from a raw examination data set and then optimize it. It also can start an optimization process from any initial solution no matter how it is generated. Actually, this is a great advantage of tabu search approach.

The OTTASYS can be used to generate a feasible solution with minimized number of timeslots for a given examination timetable, or to generate a feasible solution with a heuristically optimized number of conflicts for a given number of timeslots.

In addition, we also notice that the OTTASYS almost never performed badly compared with the results presented by Carter *et al.* This is due to the versatility of the tabu search strategy that moves exams from period to period as it attempts to minimize the objective function.

Future Work

The study on examination timetabling problems and development of the OTTASYS and its core algorithm OTTABU have provided a good starting point for further research.

A lot of research has been made on whether initial solutions have significant impact on final outcomes. Some research finds that initial solutions have a critical impact on the quality of final solutions, but some claims that their methods can correct the impact of initial solution. It seems more reasonable to assume that all heuristic methods have dependency on initial solutions. In order to test the dependency of the OTTASYS on initial solutions, more experiments can be designed and executed to test the impact of the initial solution on the optimal solutions.

We have found that the solution restructuring phase is very important and productive because it can improve solutions dramatically, but it is still not clear why and how it improves solutions. It may result from a tabu search technique, strategic oscillation, which drives solutions to cross feasible-infeasible border back and forth.

By considering that the compound moves and ejection chain approaches have proved powerful in many application areas [GloverL97] and that there are no such applications on the examination timetabling problems published [GloverL97, CarterL96, Downsland98, BurkeEF96, PATAT3], it would be an interesting research area to investigate the tabu relaxation and construction of compound move further.

Appendix A

Real Data and Solutions

Three examination timetables are presented in this appendix. Each set of exams is scheduled into 36 timeslots and each timeslot has 2200 seats at most. In order to avoid seating large numbers of students at the end of examination sessions (where there are no following exterior edges), the seating for these three timeslots is reduced to 1100.

The real data of CAR-F-92 and UTA-S-92 can be found from the URL ftp://ie.utoronto.ca/mwc/testprob. The metrics of each examination set can be found in the Table 6.1 in Chapter 6.

A1. University of Ottawa, OTT-F-96

Time slot	# of exams	1st order conflicts	2nd order conflicts	Enroll-ment	Examinations (Exam IDs)
1	9	0	5	2199	8. 405. 449. 533. 537. 659. 660. 675. 694.
2	9	0	6	510	64. 65. 233. 248. 299. 352. 418. 649. 729.
3	20	0	16	2132	11. 57. 128. 131. 161. 171. 174. 196. 211. 338. 342. 382. 401. 422. 550. 553. 669. 670. 731. 755.
4	6	0	14	624	289. 345. 362. 667. 680. 718.
5	26	0	18	2126	27. 34. 55. 91. 126. 129. 143. 151. 189. 214. 218. 250. 267. 320. 366. 394. 403. 518. 527. 534. 569. 644. 697. 701. 711. 761.

6	9	0	13	790	291, 346, 415, 464, 478, 653, 656, 665, 713,
7	26	0	12	1757	5, 19, 20, 32, 59, 70, 97, 118, 192, 198, 300, 318, 336, 386, 399, 438, 450, 519, 566, 608, 624, 626, 630, 631, 645, 700,
8	14	0	14	852	35, 71, 89, 121, 125, 212, 227, 377, 412, 504, 548, 655, 689, 742,
9	18	0	18	1115	17, 44, 81, 146, 164, 169, 281, 324, 363, 402, 417, 455, 522, 538, 568, 695, 726, 770,
10	17	0	6	1057	77, 113, 120, 175, 197, 204, 229, 283, 305, 372, 380, 411, 452, 509, 555, 666, 717,
11	11	0	15	878	4, 7, 43, 203, 298, 331, 353, 360, 390, 547, 760,
12	24	0	16	1533	15, 52, 63, 99, 145, 180, 238, 373, 421, 453, 454, 466, 482, 487, 516, 529, 561, 627, 629, 672, 676, 678, 685, 720,
13	14	0	8	1271	50, 226, 253, 271, 292, 335, 339, 349, 397, 501, 521, 636, 650, 692,
14	24	0	9	1580	18, 31, 69, 74, 76, 92, 130, 133, 311, 326, 344, 419, 483, 492, 510, 512, 524, 530, 535, 605, 617, 684, 746, 758,
15	16	0	17	979	115, 179, 209, 228, 274, 325, 431, 444, 493, 520, 592, 600, 639, 657, 698, 721,
16	34	0	22	1746	26, 37, 58, 66, 72, 93, 124, 163, 173, 182, 183, 255, 265, 288, 304, 319, 361, 388, 407, 410, 476, 484, 489, 491, 543, 562, 572, 579, 618, 643, 646, 679, 737, 767,
17	7	0	30	830	10, 137, 273, 594, 693, 710, 747,
18	36	0	6	1907	38, 107, 142, 144, 177, 185, 190, 201, 217, 224, 236, 245, 266, 287, 301, 340, 359, 375, 400, 427, 460, 465, 502, 505, 546, 573, 582, 607, 613, 654, 688, 702, 704, 706, 724, 752,

19	7	0	9	443	98. 110. 159. 474. 517. 557. 712.
20	39	0	8	1808	9. 21. 36. 45. 46. 49. 150. 176. 184. 207. 221. 231. 278. 282. 286. 302. 322. 330. 343. 371. 381. 393. 420. 424. 462. 469. 496. 508. 528. 531. 567. 578. 610. 652. 673. 682. 715. 733. 743.
21	13	0	6	838	138. 149. 222. 252. 257. 261. 432. 490. 526. 542. 606. 637. 759.
22	31	0	11	1873	14. 25. 68. 83. 84. 127. 158. 194. 210. 225. 290. 295. 309. 337. 354. 356. 365. 374. 385. 441. 511. 584. 614. 648. 662. 671. 677. 719. 739. 757. 769.
23	18	0	15	1019	103. 136. 187. 237. 242. 247. 321. 408. 414. 459. 525. 611. 628. 634. 635. 727. 736. 749.
24	33	0	15	1917	22. 23. 39. 82. 101. 112. 122. 141. 156. 199. 208. 223. 232. 234. 259. 293. 312. 333. 351. 395. 429. 430. 447. 479. 497. 507. 544. 580. 615. 642. 686. 716. 763.
25	14	0	14	874	219. 243. 254. 392. 433. 457. 475. 477. 523. 539. 552. 664. 707. 728.
26	21	0	14	1075	30. 40. 48. 60. 239. 279. 323. 369. 370. 442. 451. 514. 540. 575. 585. 609. 638. 708. 740. 751. 764.
27	32	0	10	1328	47. 88. 95. 96. 132. 135. 140. 166. 168. 181. 191. 206. 262. 307. 355. 383. 398. 425. 435. 445. 472. 481. 488. 495. 556. 593. 603. 658. 661. 674. 696. 744.
28	25	0	16	1289	28. 41. 54. 62. 108. 213. 244. 272. 275. 294. 368. 389. 413. 456. 545. 554. 570. 596. 619. 622. 625. 714. 735. 754. 766.

29	34	0	7	2193	2, 3, 12, 73, 79, 87, 116, 117, 119, 134, 155, 160, 241, 269, 306, 313, 357, 384, 404, 406, 436, 467, 485, 494, 560, 586, 588, 598, 633, 647, 681, 683, 699, 703,
30	24	0	30	1088	51, 67, 80, 100, 251, 264, 270, 276, 277, 296, 317, 328, 334, 348, 379, 513, 536, 549, 577, 595, 640, 723, 734, 745,
31	26	0	19	1468	16, 75, 111, 139, 157, 172, 186, 188, 205, 216, 391, 409, 434, 470, 471, 473, 499, 532, 558, 564, 663, 691, 750, 756, 765, 768,
32	26	0	12	1005	90, 94, 102, 104, 123, 178, 249, 258, 280, 284, 285, 297, 315, 329, 347, 358, 364, 423, 506, 541, 576, 587, 590, 591, 612, 690,
33	41	0	16	1514	1, 13, 29, 53, 56, 61, 85, 114, 153, 165, 193, 200, 202, 215, 235, 240, 263, 268, 376, 437, 439, 440, 446, 458, 463, 480, 498, 559, 565, 571, 589, 604, 620, 621, 623, 668, 705, 722, 730, 748, 762,
34	38	0	9	1085	24, 78, 105, 106, 109, 147, 152, 162, 220, 303, 308, 310, 314, 327, 341, 350, 367, 387, 416, 428, 443, 448, 468, 486, 551, 563, 581, 597, 599, 616, 632, 641, 687, 725, 738, 741, 753, 771,
35	22	0	1	1098	6, 33, 42, 86, 148, 154, 167, 170, 195, 230, 260, 316, 378, 396, 426, 500, 503, 515, 574, 583, 602, 651,
36	7	0	0	1098	246, 256, 332, 461, 601, 709, 732,
Total	771	0	457	46899	

A2. Carleton University, CAR-F-92

Time slot	# of exams	1st order conflicts	2nd order conflicts	Enroll -ment	Examinations (Exam IDs)
1	3	0	46	2200	185, 204, 205,
2	16	0	43	1345	15, 24, 68, 121, 152, 158, 166, 212, 267, 269, 338, 382, 453, 455, 465, 492,
3	8	0	96	1315	74, 189, 213, 221, 236, 301, 352, 432,
4	13	0	44	1551	49, 92, 93, 105, 136, 165, 167, 178, 201, 282, 372, 420, 509,
5	14	0	55	1704	45, 60, 115, 151, 179, 190, 242, 313, 354, 422, 428, 429, 436, 442,
6	12	0	171	1872	6, 70, 202, 210, 224, 249, 278, 329, 331, 332, 345, 506,
7	9	0	154	1498	101, 188, 270, 288, 302, 326, 361, 430, 460,
8	12	0	123	2068	7, 109, 122, 138, 207, 322, 368, 376, 389, 401, 449, 498,
9	12	0	69	1394	44, 73, 118, 146, 197, 239, 251, 315, 367, 400, 423, 515,
10	9	0	33	1348	1, 65, 90, 116, 173, 306, 371, 396, 443,
11	14	0	22	1135	21, 85, 87, 135, 198, 220, 261, 280, 323, 342, 399, 438, 458, 538,
12	18	0	103	2163	61, 113, 245, 246, 255, 256, 257, 258, 259, 260, 317, 334, 341, 350, 373, 385, 391, 490,
13	16	0	67	1539	104, 144, 145, 159, 180, 279, 285, 303, 304, 355, 394, 440, 459, 481, 508, 522,
14	14	0	80	1498	71, 79, 124, 142, 161, 223, 344, 362, 384, 402, 467, 470, 489, 516,

15	18	0	74	1870	53, 94, 120, 133, 134, 150, 155, 177, 195, 209, 300, 312, 340, 379, 381, 392, 448, 527,
16	18	0	58	1359	2, 54, 55, 83, 97, 194, 215, 265, 375, 414, 471, 482, 510, 517, 519, 521, 532, 543,
17	11	0	55	1806	9, 110, 132, 191, 266, 283, 289, 290, 298, 299, 405,
18	14	0	31	1277	31, 56, 77, 108, 141, 319, 343, 378, 409, 418, 441, 480, 503, 542,
19	19	0	39	1459	4, 20, 80, 86, 112, 130, 140, 182, 183, 219, 237, 281, 308, 335, 336, 369, 387, 427, 445,
20	19	0	22	1669	29, 33, 34, 52, 62, 67, 160, 241, 250, 346, 353, 390, 412, 426, 457, 486, 501, 533, 539,
21	18	0	44	1253	8, 27, 32, 84, 117, 139, 184, 203, 226, 262, 277, 287, 318, 325, 419, 425, 437, 479,
22	17	0	38	1594	30, 36, 47, 126, 131, 148, 174, 238, 244, 347, 365, 411, 504, 505, 511, 536, 541,
23	13	0	45	1158	50, 89, 95, 99, 137, 164, 181, 200, 225, 227, 348, 404, 417,
24	22	0	50	2192	11, 12, 114, 153, 163, 176, 206, 208, 243, 275, 311, 351, 370, 374, 408, 435, 466, 476, 477, 499, 513, 524,
25	13	0	36	977	19, 42, 43, 51, 63, 72, 147, 149, 192, 268, 333, 456, 464,
26	28	0	71	2121	38, 39, 40, 41, 46, 48, 88, 107, 168, 172, 175, 193, 196, 222, 264, 276, 324, 366, 377, 393, 439, 446, 454, 469, 478, 485, 537, 540,
27	8	0	35	1216	17, 18, 35, 78, 156, 330, 337, 468,
28	25	0	49	1759	37, 64, 125, 230, 235, 248, 271, 272, 273, 305, 310, 314, 320, 357, 397, 398, 403, 416, 444, 447, 487, 494, 500, 530, 535,

29	15	0	55	1102	14, 75, 96, 103, 128, 154, 171, 216, 254, 360, 386, 388, 406, 431, 526,
30	20	0	28	1541	5, 16, 81, 102, 232, 247, 263, 274, 321, 349, 358, 380, 462, 475, 484, 491, 493, 502, 523, 525,
31	25	0	21	1711	13, 25, 26, 28, 57, 66, 76, 98, 129, 186, 199, 214, 218, 229, 286, 309, 339, 359, 383, 407, 421, 472, 474, 514, 531,
32	19	0	52	1482	82, 100, 119, 143, 252, 284, 292, 293, 294, 295, 296, 327, 364, 415, 451, 463, 488, 497, 529,
33	17	0	51	2078	10, 22, 69, 91, 169, 187, 231, 233, 253, 291, 307, 395, 450, 473, 483, 507, 512,
34	22	0	15	1072	23, 58, 59, 106, 111, 127, 162, 211, 217, 234, 240, 316, 328, 363, 413, 424, 461, 496, 518, 520, 528, 534,
35	10	0	17	1097	3, 123, 157, 170, 228, 297, 410, 433, 434, 495,
36	2	0	0	1099	356, 452,
Total	543	0	1992	55522	

A3. University of Toronto, UTA-S-92

Time slot	# of exams	1st order conflicts	2nd order conflicts	Enroll-ment	Examinations (Exam IDs)
1	5	0	91	2198	15, 22, 123, 158, 336,
2	15	0	86	1386	76, 108, 144, 173, 188, 219, 225, 247, 305, 378, 439, 441, 468, 491, 529,
3	14	0	141	2009	4, 25, 47, 55, 98, 103, 113, 135, 286, 309, 404, 420, 460, 508,
4	14	0	126	2074	89, 107, 151, 217, 261, 333, 388, 501, 513, 518, 528, 584, 589, 622,
5	21	0	690	2118	26, 54, 65, 80, 92, 109, 206, 272, 274, 402, 412, 423, 442, 464, 465, 483, 490, 494, 553, 593, 599,
6	14	0	65	2070	17, 29, 134, 164, 251, 277, 285, 300, 386, 391, 428, 542, 543, 567,
7	13	0	43	1815	1, 10, 34, 178, 216, 270, 409, 434, 446, 466, 473, 511, 578,
8	18	0	49	1397	20, 81, 83, 111, 122, 139, 182, 185, 197, 213, 238, 262, 271, 375, 429, 452, 476, 545,
9	13	0	87	1715	5, 23, 106, 133, 138, 268, 278, 365, 393, 462, 495, 525, 606,
10	18	0	79	1981	3, 66, 78, 126, 146, 165, 174, 186, 201, 260, 282, 304, 316, 399, 408, 430, 576, 614,
11	9	0	80	1389	43, 85, 121, 332, 438, 449, 482, 504, 527,

12	22	0	55	1824	56, 97, 99, 137, 156, 170, 207, 231, 267, 279, 321, 322, 347, 369, 372, 373, 418, 436, 550, 598, 603, 608,
13	16	0	77	1424	6, 101, 120, 166, 167, 191, 192, 234, 246, 313, 356, 385, 425, 453, 467, 541,
14	13	0	75	1634	37, 61, 62, 155, 248, 263, 284, 437, 515, 565, 573, 583, 595,
15	19	0	48	1589	74, 112, 169, 189, 296, 315, 325, 342, 367, 381, 383, 384, 387, 427, 448, 486, 577, 604, 618,
16	18	0	52	1536	91, 160, 212, 227, 240, 281, 314, 355, 361, 406, 407, 454, 458, 523, 555, 568, 600, 607,
17	16	0	45	1258	11, 18, 33, 73, 94, 142, 153, 176, 183, 198, 255, 293, 328, 447, 594, 612,
18	19	0	46	1408	27, 68, 147, 187, 194, 242, 256, 307, 345, 374, 398, 419, 435, 469, 487, 521, 539, 591, 596,
19	26	0	49	1849	30, 31, 39, 69, 88, 127, 136, 157, 180, 184, 204, 210, 222, 241, 338, 368, 379, 459, 488, 505, 536, 537, 546, 570, 571, 586,
20	15	0	51	1166	7, 53, 70, 253, 306, 311, 340, 397, 445, 463, 524, 549, 554, 587, 609,
21	19	0	78	1639	35, 86, 115, 150, 161, 190, 196, 244, 257, 259, 283, 302, 380, 424, 484, 493, 509, 516, 597,
22	18	0	27	1826	75, 159, 200, 339, 351, 357, 358, 359, 360, 364, 396, 474, 492, 517, 530, 535, 562, 613,

23	13	0	39	2026	110, 230, 249, 269, 273, 287, 299, 301, 352, 354, 403, 461, 601,
24	20	0	48	1422	2, 19, 79, 95, 124, 140, 171, 209, 226, 235, 289, 324, 329, 400, 421, 422, 471, 526, 581, 611,
25	20	0	54	1662	21, 32, 48, 51, 71, 90, 116, 118, 202, 290, 317, 410, 414, 440, 451, 470, 479, 496, 506, 559,
26	16	0	38	1357	64, 130, 152, 177, 195, 245, 252, 295, 312, 331, 344, 431, 450, 457, 500, 552,
27	18	0	54	1464	9, 24, 45, 93, 105, 131, 205, 211, 218, 280, 319, 341, 366, 377, 395, 503, 582, 592,
28	20	0	62	1538	59, 128, 132, 193, 229, 237, 243, 294, 298, 323, 350, 371, 382, 390, 477, 510, 514, 531, 563, 575,
29	24	0	61	1770	13, 38, 49, 82, 87, 114, 148, 215, 221, 223, 232, 275, 291, 362, 498, 507, 522, 551, 569, 590, 602, 610, 616, 620,
30	18	0	84	1312	8, 50, 60, 63, 125, 175, 228, 239, 264, 334, 346, 394, 415, 475, 481, 502, 556, 579,
31	21	0	49	2186	12, 28, 36, 44, 57, 67, 84, 141, 149, 168, 303, 320, 326, 349, 433, 478, 489, 544, 548, 574, 605,
32	23	0	28	1447	16, 77, 102, 143, 163, 181, 220, 236, 265, 266, 276, 327, 401, 405, 417, 426, 443, 456, 499, 532, 558, 561, 617,

33	25	0	11	2199	41, 96, 104, 145, 172, 203, 308, 310, 318, 337, 348, 353, 363, 370, 376, 389, 411, 413, 444, 512, 520, 538, 566, 585, 621,
34	19	0	22	1098	58, 100, 117, 129, 154, 162, 208, 214, 224, 233, 254, 288, 292, 480, 497, 534, 540, 580, 615,
35	12	0	10	1093	14, 179, 199, 258, 297, 343, 392, 432, 547, 564, 588, 619,
36	18	0	0	1100	40, 42, 46, 52, 72, 119, 250, 330, 335, 416, 455, 472, 485, 519, 533, 557, 560, 572,
Total	622	0	2700	58979	

References

Papers and Books

[BoufletS96] Jean P. Boufflet and Stephen Negre, *Three Methods Used to Solve an Examination Timetabling Problem*, LNCS1153, P.327-334.

[Bulln98] Bernd Bullnheimer, *An Examination Scheduling Model to Maximize Student's Study Time*, LNCS1408, 1998, P.78-91.

[BurkeEF96] Edmund K. Burke, D. G. Elliman, P. H. Ford, and R. F. Weare, *Examination Timetabling in British Universities: A Survey*, LNCS1153, 1996, P. 76-92.

[BurkeNW96] Edmund K. Burke, J. P. Newall, and R. F. Weare, *A Memetic Algorithm for University Exam Timetabling*, LNCS1153, 1996, P. 241-250.

[CarterL96] Mike W. Carter and Gilbert Laporte, *Recent Developments in Practical Examination Timetabling*, LNCS1151, 1996, P.3-21.

[CarterLC94] Mike W. Carter, Gilbert Laporte and J. W. Chineck, *A General Examination Scheduling System*, INTF, vol.24, May-Jun, 1994, P.109-120.

[CarterLL96] Mike W. Carter, Gilbert Laporte and Sau Yan Lee, *Examination Timetabling: Algorithmic Strategies and Applications*, JORS, Vol.47, 1996, P.373-383.

[Carter86] Mike W. Carter, *A Survey of Practical Application of Examination Timetabling Algorithm*, OR, Vol. 34, No. 2, March-April, 1986, P.193-202.

[CorneR96] David Corne and Peter Rose, *Peckish Initialization Strategies for Evolutionary Timetabling*, LNCS1153, 1996, P.227-240.

[CostaH97] Daniel Costa and Alain Hertz, *Ants Can Colour Graphs*, JORS, Vol. 48, 1997, P.295-305.

[Costa94] Daniel Costa, *A Tabu Search Algorithm for Computing An Operational Timetable*, EJOR, Vol. 76, 1994, P.98-110.

[Downsland98] Kathryn A. Downsland, *Off-the-Peg or Made-to-Measure? Timetabling and Scheduling with SA and TS*, LNCS1408, 1998, P.37-52.

[Ferland98] Jacques A. Ferland, *Generalized Assignment-Type Problems: A Powerful Modeling Scheme*, LNCS1408, 1998, P.53-77.

[GasperoS00] Luca Di Gaspero and Andrea Schaerf, *Tabu Search Techniques for Examination Timetabling* (abstract), PATAT3, P. 176-179.

[GloverL97] Fred Glover and M. Laguna, *Tabu Search*, Kluwar Academic Publishers, 1997

[Glover90] Fred Glover, *Tabu Search: A Tutorial*, INTF, July-August, 1990, P.74-94.

[HertzW87] Alain Hertz and Dominique de Werra, *Using Tabu Search Techniques for Graph Coloring*, COMP, Vol. 39, 1987, P.345-351.

[Hertz91] Alain Hertz, *Tabu Search For Large Scale Timetabling Problems*, EJOR, Vol. 54, 1991, P.39-47.

[Newell99] J. P. Newell, *Hybrid Methods for Automated Timetabling*, A Ph.D Dissertation, Nottingham University, UK, 1999.

[PeckW66] J.E.L. Peck and M. R. Williams, *Algorithm 286 – Examination Scheduling*, Comm. ACM., vol. 9, 1966, p.433-434

[RoseCF94] Peter Rose, Dave Corne, and Hsiao-Lan Fang, *Improving Evolutionary Timetabling With delta Evaluation and Directed Mutation*, PPSN3, 1994, P.565-566.

[RoseHC98] Peter Rose, Emma Hart, and Dave Corne, *Some Observations about GA-Based Exam Timetabling*, LNCS1408, 1998, P.115-129.

[Werra85] Dominique de Werra, *An Introduction to Timetabling*, EJOR, Vol. 19, 1985, P.151-308.

[Werra96] Dominique de Werra, *Some Combinatorial Models for Course Scheduling*, LNCS1153, 1996, P.296-308.

[WhiteB00] George M. White and B. S. Xie, *Examination Timetables and Tabu Search With Longer Term Memory*, PATAT3, page 184-201.

[WhiteC79] George M. White and P. W. Chan, *Towards the Construction of Optimal Examination Schedules*, INFOR, Vol. 17, 1979, P.219-229.

[WhiteH83] George M. White and M. Haddad, *A Heuristic Method for Optimizing Examination Schedules Which Have Day and Night Courses*, Computer and Education, Vol. 7, 1983, P.235-238.

[WhiteZ98] George M. White and Junhan Zhang, *Generating Complete University Timetables by Combining Tabu Search with Constraint Logic*, LNCS1408, 1998, P.187-198.

Abbreviations of Magazines and Conferences

[COMPG] QA76.C582, *Computing*.

[EJOR] T57.6.E92, *European Journal of Operational Research*.

[INTF] *Interfaces*.

[JORS] Q175, *Journal of the Operational Research Society*.

[LNCS1153] QA76.L42. Lecture Notes In Computer Science, V1153, Springer-Verlag, Berlin, 1996.

[LNCS1408] QA76.L42. Lecture Notes In Computer Science, V1408, Springer-Verlag, Berlin, 1998.

[OR] Q175, *Operations Research.*

[PATAT3] PATAT 2000, *Proceedings of the 3rd International Conference on the Practice and Theory of Automated Timetabling*, August 16-18, 2000, Konstanz, Germany

[PPSN3] Y. Davidor, H. P. Schwefel, and R. Manner (editors), *Parallel Problem Solving in Nature, Vol. 3*, Springer-Verlag, Berlin, 1994