MULTIPLE KNAPSACK PROBLEM WITH INTER-RELATED ITEMS AND ITS APPLICATIONS TO REAL WORLD PROBLEMS

ANG JUAY CHIN

(B.Sc (Computer and Information Sciences), NUS)

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

NATIONAL UNIVERSITY OF SINGAPORE

2003

Abstract

Motivated by a real world application, we study a variant of the classic knapsack problem, which we call the Multiple Knapsack Problem with Inter-related Items (MKPIR). We are given a set of items and a set of knapsacks of limited capacity. For each item, a set of knapsacks that can hold that item is specified. In addition, binary relationships may exist between the items. These relationships affect the profit of having that item in the knapsack in consideration. In this thesis, we adopted a few heuristics and test them on the venue assignment problem, instance of MKPIR, using actual data and randomly generated test instances.

Keywords:

Multiple knapsack problem, Venue assignment problem

Acknowledgements

I would like to express my sincere gratitude to my supervisor A/P Andrew Lim for his sound advice through the course of my study. I would also like to thank my co-supervisor Dr Tan Sun Teck for his guidance.

Lastly, I would like to express my heart felt love and appreciation to my family especially my husband Wee Kit for their support, patience and love during this period.

Table of Contents

List of Figures

List of Tables

Summary

In the Multiple Knapsack Problem, N items of different sizes have to be packed into M knapsacks with limited volume. Each item i has an associated profit p_i and weight w_i. The problem is to select N disjoint subsets of items, such that subset i fits into knapsack i and total profit of the selected items is maximized.

Motivated by a real world application, we study a variant of MKP, the Multiple Knapsack Problem with Inter-related Items (MKPIR). In MKPIR, binary relationships may exist between items. Consider two items, i and j, packed in knapsacks x and y respectively. If there exists a relationship r_{ij} between the two items, then the value of the relationship between i and j is $r_{ij} \cdot k_{xy}$ where k_{xy} is the *distance measure* between knapsacks x and y. Accordingly, item i has profit $p_{ix} = \sum_{j=1}^{N} r_{ij} k_{xy}$ when considered for inclusion in knapsack x. Thus, unlike most other variants of MKP in which the profit/cost of item i is fixed, p_i, the value of item i in MKPIR is a variable. In addition, an assignment restriction is imposed on the items. For each item i, a set of knapsacks that can hold item i is specified. In a feasible assignment of items to knapsacks, each item is assigned to at most one knapsack, assignment restrictions are satisfied, and knapsack capacities are not exceeded, with the objective of maximizing $\sum_{x=1}^{M}\sum_{i=1}^{N} p_{ix} z_{ix}$, where $z_{ix} = 1$ if item i is assigned to knapsack x and $z_{ix} = 0$ otherwise.

The *Venue Assignment Problem* (VAP) is a sub-problem in an exam-timetable scheduling application. In this application, examinations for modules are assigned exam timeslots based on the estimated enrolment figures, disregarding the actual

examination venue. Specific venues are only allocated when the actual enrolment figures for modules are known after student registration. Each day of the examination period is divided into a few non-overlapping exam sessions. Venues may be physically far apart, and some candidates may be required to take more than one exam a day and have to travel between venues. Thus, in addition to assigning examinations to venues without violating the venue capacity constraint, we would like to minimize the distance that candidates have to travel between their examinations each day.

The problem of assigning exams to venues so as to minimize student movement can be formulated as an instance of MIN_MKPIR (a minimization version of MKPIR). The assignment of exams to each exam timeslot is restricted to only venues available during that timeslot. The distance that needs to be traveled by candidates of exam i to his next exam j on the same day will be translated into cost for both exam i and exam j. Each exam day may be viewed as a separate VAP, since exams scheduled on different days are independent.

Various heuristics such as Tabu Search, Simulated Annealing and "Squeaky Wheel Optimization" are experimented on the MKPIR using actual data from the National University of Singapore and generated test data. Preliminary results based on our experimentations shows that Simulated Annealing and "Squeaky Wheel Optimization" with Tabu Search post-optimization produce satisfactory results. Although more work still need to be done on solving the MKPIR efficiently, we believe that our heuristics will be useful for solving VAP and other real world problems of similar characteristics.

Chapter 1

Introduction

1.1 Motivation

Our variant of the multiple knapsack problem, *multiple knapsack problem with inter-related items* was motivated by a sub-problem in the timetabling problem in the National University of Singapore (NUS). NUS conducts more than 1500 modules for about 25 000 students every semester. In 1993, a modular course system was adopted by NUS. Under this modular system, students progress at their own pace and choose the modules that they wished to study, subject to timetable arrangements, in order to complete their degree requirements. In addition, students can choose from a wide range of modules, called cross-faculty modules, offered by different faculties. As a result of this flexibility for students, the task of scheduling examination timetables in NUS became much more complex.

Previously, the scheduling of examination timetable in NUS was done manually. This was a tedious and error-prone process that typically took months to process. Critical conflicts where a student was scheduled to take examinations at the same time were not always detected, and correcting these mistakes was cumbersome and expensive. A team was thus funded by NUS to develop an automated campus-wide examination timetabling system, *UTTSExam* [Lim et al. 2000].

Scheduling of examinations in *UTTSExam* is a two steps process. Examinations for modules are first assigned timeslots based on estimated enrolments without venues. Specific venues are only assigned in the second phase when actual enrolment figures are known. This created the venue assignment problem and motivates the study of the *multiple knapsack problem with inter-related items* in this report.

1.2 Examination Timetabling Problem

The primary objective of an examination timetabling problem is to schedule all examinations into available timeslots without hard constraints violation. Many practical examination timetabling problems are however much more complex and usually include the assignment of examinations to various venues of varying capacities and the satisfying of various timetable constraints. These timetable constraints may be hard or soft constraints. The satisfaction of hard constraints is considered essential to a feasible timetable, whereas the satisfaction of soft constraints is considered desirable but not essential. A high quality timetable is one that meets all the conditions of a feasible timetable and minimizes the violation of these soft constraints.

It is the University's policy to schedule all examinations before student enrolment. The automated examination timetable scheduler hence has to schedule examinations based on estimated enrolment figures put forward by the various faculties. These enrolment estimates are however not accurate and are often under-estimated or over-estimated. As such, to better utilize available resources, specific venues are only assigned after student registration.

A hybrid of centralized and de-centralized approach is adopted for timetabling in *UTTSExam* [Lim et al. 2002]. Scheduling of examination timetable is carried out in two phases. In the first phase, before student registration, the examination's enrolment estimates were used to proportionally allocate a fixed number of *seats* for each faculty (called their *venue partition*). Each faculty then scheduled their exams according to their venue partition, disregarding the actual venues. These faculty timetables are then merged into a campus-wide tentative timetable. The central authority will resolve any conflicts that arise during the merging that are caused by modules offered to more than one faculty. Specific venues are only allocated in the second phase when the actual enrolment figures for modules are known after student registration. This approach proved to create a much better timetable than a fully centralized approach. Furthermore, it allows the individual faculty timetable administrators to retain a high level of control over their personal timetables, which would not have been possible in a fully centralized system.

1.3 Venue Assignment Problem

The venue assignment problem (VAP) is a sub-problem of the examination timetabling problem in *UTTSExam.* The objective is to assign a set of examinations with known timeslots to venues with limited capacities while minimizing travelling required by candidates who have back-to-back examinations.

Each day of the examination period is divided into a few non-overlapping examination sessions. Due to the limited amount of spaces suitable to hold examinations on-campus, off-campus venues have to be included. Examination venues

thus may be physically far apart. Candidates who are required to take more than one examination a day may thus have to travel between on-campus venues and off-campus venues.

Starting 2002/2003 academic year, NUS decided to do away with commercial venues for examinations. Instead, multiple small rooms located around campus are made available to make up the deficit. Though this has resulted in a substantial monetary savings in rental cost, candidates with back-to-back examinations are not spared from having to travel between far apart examination venues within the campus. As individual faculties do not have enough venue capacity to hold all examinations conducted by their own faculties, central venues have to be used. In addition, due to the existence of cross-faculty modules which is taken by students from different faculties, it is inevitable that some candidates may have to travel between examination venues.

It is the University policy that examination timetable is planned and released before student registration so that students have a greater control over their examination schedule. Since, student registration data is not available during timetable scheduling, it is therefore not within the examination timetable scheduler control to minimize the number of back-to-back examinations. The best that the examination timetable planner can do is to update the automatic timetable scheduler with constraints that ensure that certain examinations are placed at least a number of sessions apart. These constraints may be based on popular module combinations taken by past year students. The scheduler will avoid planning examinations of these

modules on back-to-back sessions. All these are however just predictions and will not guarantee that the number of back-to-back examinations for students is minimized.

Under these circumstances, it is important that the number of candidates that needs to travel between examination venues be kept to the minimum, otherwise traffic control will be chaotic during the exam session interval. It is therefore one of the objectives in VAP to minimize *student movement*; that is the distance that candidates have to travel between back-to-back examinations each day.

Additionally, assignment restrictions, which we call *venue constraints* in VAP, must not be violated, since some examinations must be held in a specific type of venue. For instance, some examinations must be held in drawing rooms with big tables since large drawing papers are involved during the examination, while other papers should not be held in the drawing rooms.

Before, the assignment of venues to examinations was done manually in the second phase of the examination timetable planning exercise, weeks before the start of the examination period. The assignment of venues to examinations has to be carefully managed manually. Student movement has to be painstakingly calculated and venue constraints have to be satisfied. A careless assignment that results in massive group of students having to travel between far-away venues not only causes great inconvenience to students but will also impose a heavy burden on the school transportation system.

We see that the venue assignment problem plays an important part in the goodness of the examination timetable in *UTTSExam*. Since no matter how good the examination timetable is in the first phase of the scheduling process, a bad venue assignment in the second phase will eventually lead to an overall poor solution. A

venue assignment that results in many candidates having to travel between far-away venues for exams on the same day is deemed to be a poor assignment.

In the event of an emergency (for example, Severe Acute Respiratory Syndrome (SARS) outbreak) when the University needs to re-assign examinations to venues within a very short period of time, a venue assignment engine will be very critical and handy. This was the situation for examinations in semester 2 2002/2003 when SARS outbreak occur just a few weeks before the examination period. As SARS can be spread by droplet infection and close contact with SARS patients, the university has to make re-arrangements for examinations to limit exposure and risk (http://www.nus.edu.sg/sars/measures.htm). This includes reducing the size of groups by de-centralizing exams to many venues and increasing the table to table distance in venues. The increase in table to table distance caused a drop of more than 20% in the number of seats available in the examination venues used. For example, for one of the major exam venue, Multi-Purpose Hall 1, the number of seats available decreases from 850 to 660. The availability of a venue assignment engine could quickly perform a re-assignment and advise if more venues are required.

We show that the problem of assigning examinations to venues can be modeled as a multiple knapsack problem in chapter 2.

1.4 Thesis Outline

In this thesis, we study the Multiple Knapsack Problems with Inter-related Items (MKPIR), which is a variant of MKP, in which binary relationships exist between items and adopted a few heuristics for solving knapsack problems of this

variant. Practical applications for MKPIR include the venue assignment problem and shelf space allocation problem (SSAP)[Yang 2001]. Chapter 2 explains the MKPIR in details and models the VAP as an instance of MKPIR. Chapter 3 shows some interesting user interfaces and key functionalities offered by the *UTTSExam* System. In Chapter 4, various heuristics are presented. They include Tabu Search, Simulated Annealing and "Squeaky Wheel Optimization". We have attempted to post-optimized the results from "Squeaky Wheel Optimization" using Tabu Search. In Chapter 5, we demonstrate the effectiveness of our heuristics by studying its use on VAP, a subproblem of an examination-scheduling problem, based on real data from the National University of Singapore and generated test instances. The details of the experiments performed using these heuristics and hybrids are given and our experimental results are analyzed. Finally, in Chapter 6 we discuss possible future works and present our conclusions.

Chapter 2

Multiple Knapsack Problem

Knapsack Problems have been intensively studied since the emergence of Combinatorial Optimization due to their many applications in industry and financial management. They also appear as a sub-problem in many more complex algorithms, and these algorithms will benefit from any improvement in this field. The family of knapsack problems all considers a set of items, each associated with a profit and weight. The objective is to choose a subset of the given items such that the corresponding profit sum is maximized without exceeding the capacity of the knapsack(s). Different types of Knapsack Problems arise depending on the distribution of items and knapsacks and the constraints involved.

In the multiple knapsack problem (MKP)[Martello and Toth, 1990; Pisinger and Toth 1998], *N* items of different sizes have to be packed into *M* knapsacks with limited volume. Each item *j* has an associated profit p_j and weight w_j. The problem is to select *M* disjoint subsets of items, such that subset *i* fits into knapsack *i* and total profit of the selected items is maximized.

In this report, we study the Multiple Knapsack Problems with Inter-related Items, a variant of MKP, in which binary relationships exist between items. The next section shall describe the problem in detail.

2.1 Multiple Knapsack Problem with Inter-related Items

In Multiple Knapsack Problems with Inter-related Items (MKPIR), we are given a set of N items to be packed into a set U of M knapsacks. Each item i has a positive weight w_i, and each knapsack x has a limited capacity v_x. In addition, binary relationships may exist between items. Consider two items, i and j, packed in knapsacks x and y respectively. If there exists a relationship r_{ij} between the two items, then the value of the relationship between i and j, is $r_{ij}k_{xy}$ where k_{xy} is the *distance measure* between knapsacks x and y. Accordingly, item i has profit $p_{ix} = \sum_{j=1}^{N} r_{ij}k_{xy}$ when considered for possible inclusion in knapsack x. Thus, unlike most other variants of MKP in which the profit of item i is a constant, the value of item i in MKPIR, p_{ix} varies. The value of p_{ix} depends on the relationship of item i with other items and the knapsack solution. In addition, an assignment restriction is imposed on the items. For each item i, a set $A_i \subseteq U$ of knapsacks that can hold item i is specified. The objective is to

$$\text{maximize} \qquad P = \sum_{x=1}^{M}\sum_{i=1}^{N} p_{ix} z_{ix} \qquad (1)$$

$$\text{subject to} \qquad \sum_{i=1}^{N} w_i z_{ix} \le v_x \ , \quad x = 1,...,M \ . \qquad (2)$$

where $z_{ix} = 1$ if item i is assigned to knapsack x and $z_{ix} = 0$ otherwise

and that assignment restrictions are satisfied.

The minimization version of the problem MIN_MKPIR is obtained by defining c_{ix} as the cost required to assign item i to knapsack x. However, to simply replace p_{ix} by c_{ix} and state the objective of MIN_MKPIR as minimize $\sum_{x=1}^{M}\sum_{i=1}^{N} c_{ix} z_{ix}$ alone allows for the trivial solution with 0 cost where no items are assigned to any knapsack. Hence MIN_MKPIR is defined as

$$\text{minimize} \quad \sum_{x=1}^{M}\sum_{i=1}^{N} c_{ix} z_{ix} \tag{3}$$

$$\text{subject to} \quad \sum_{i=1}^{N} w_i z_{ix} \le v_x \ , \qquad x = 1,..., M \ . \tag{4}$$

where $z_{ix} = 1$ if item i is assigned to knapsack x and $z_{ix} = 0$ otherwise

while maximizing $\sum_{i=1}^{N} w_i z_{ix}$ where $x = 1,..., M$ and that assignment restrictions are satisfied. Section 5.2 will discuss in detail how we evaluated these objectives in MKPIR.

2.2 Formulation of VAP as MIN_MKPIR

The problem of assigning examinations to venues so as to minimize travel distance for candidates, subjected to assignment restriction, can be formulated as an instance of the MIN_MKPIR.

Each examination day may be viewed as a separate VAP, since examinations scheduled on different days are independent. It is not a concern if candidates have to

take an examination in venue x on one day and another examination in venue y the next day.

In VAP, item refer to examination and the number of candidates for examination i, which we also refer to as size of examination i is denoted by w_i. Due to the nature of the problem, all candidates for each examination i must be assigned an examination venue. The assignment of examinations to venues in each examination timeslot is restricted to only the venues applicable during that timeslot. For each item i, A_i is the set of venues applicable in the timeslot in which examination i is scheduled, while taking its *venue constraint* into consideration.

The distance that needs to be traveled by candidates of examination i to examination j which is scheduled on consecutive timeslots on the same day is translated into cost for both examinations i and j. The number of candidates that need to sit for both examinations i and j, $1 \le i, j \le M$, is represented by r_{ij}. If examination i is held in venue x while examination j is held in venue y, then k_{xy} is the distance between venue x and venue y. Hence r_{ij} candidates will need to travel k_{xy} distance if examination i is held in venue x while examination j is held in venue y. Cost of holding examination i in venue x is thus $c_{ix} = \sum_{j=1}^{N} r_{ij} \cdot k_{xy}$.

A good solution to the venue assignment problem is one with minimum cost $\sum_{x=1}^{M}\sum_{i=1}^{N} c_{ix} z_{ix}$ and that packed as many exams as possible without splitting across multiple venues.

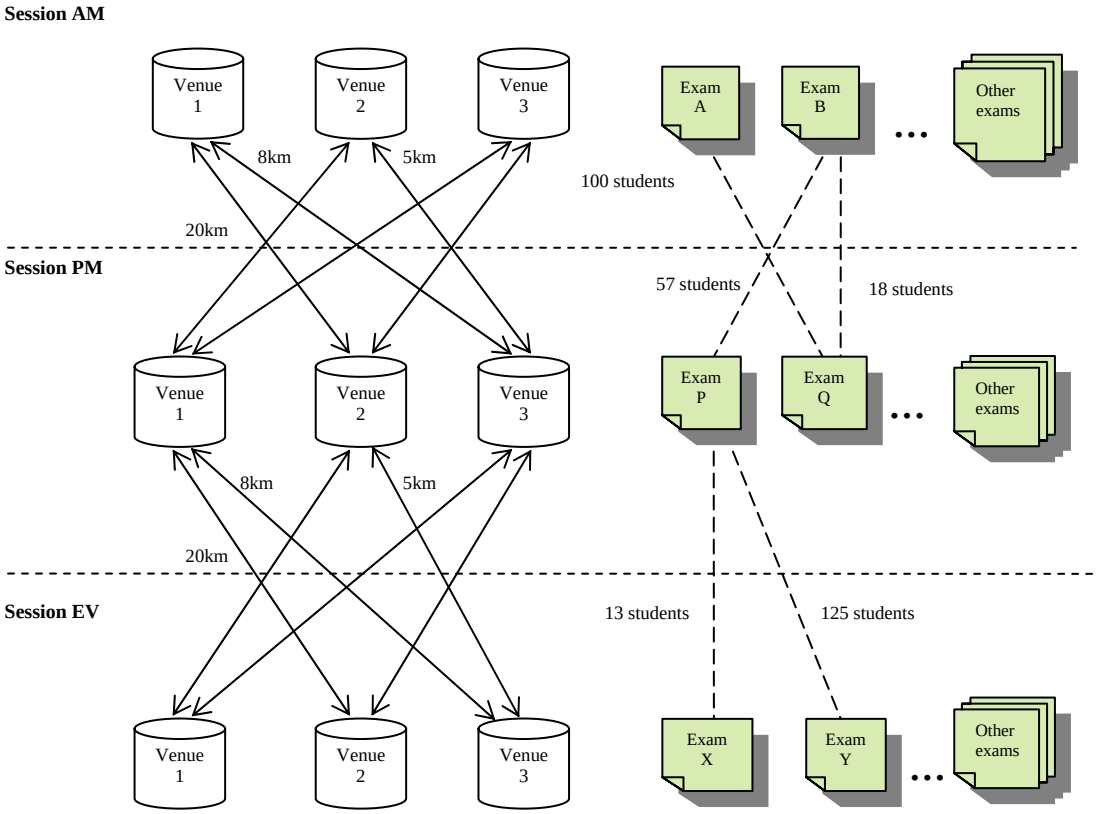

Figure 1: Venue Assignment Problem as an instance of a MKPIR

Figure 1 shows a venue assignment problem for an examination day. The number of candidates that have to sit for back-to-back examinations is shown along the dotted line. The distance k_{xy} between two venues is shown on the solid line. For example, if there are 57 students who take both examination B and examination P ($r_{BP} = 57$), and examination B is assigned to venue 1 while examination P is assigned to venue 2, then 57 students will have to travel from venue 1 to venue 2 for their next examination. Cost of holding examination B in venue 1 will be 57×20 since the distance between venue 1 and venue 2, $k_{12} = 20$.

2.3 Related Work

MKP is NP-hard in the strong sense [Martello and Toth 1990] and thus no fully polynomial approximation scheme can be found unless $P = NP$ [Garey and Johnson 1979]. Several exact algorithms and approximation algorithms have been presented to solve the multiple knapsack problem.

[Hung and Fisk 1978] presented a depth-first branch and bound algorithm where the upper bounds were derived by either Lagrangian relaxation or surrogate relaxation. Choice of the item selected at each level of the decision then depends on the relaxation used. In the Lagrangian case, the algorithm selects the item which had been inserted in the most knapsacks. While in the surrogate case, among all the items that are not assigned, the item with the lowest index is selected. The items are sorted according to $\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq ... \geq \frac{p_N}{w_N}$ and the knapsacks are sorted in non-increasing order.

Previous work on MKP and its variants [Shachnai and Tamir, 2001][Dawande and Kalangnanam 1998] however assume that (i) items are not related and that relationship between items has no effect on the assignment, (ii) profit/cost of items has a fixed value, regardless of the knapsack it is being assigned to and its relationship with other items, and (iii) items may be assigned to any knapsack as long as the capacity of the knapsack is not exceeded. Though $1/2$-approximation algorithms were presented in [Dawande et al. 2000] for a MKP with assignment restriction, profit/cost of items is however fixed and there is no relationship between the items.

Chapter 3

System User Interface

This chapter shows some of the key functions and screenshots of the UTTSExam System.

3.1 Exam Scheduler

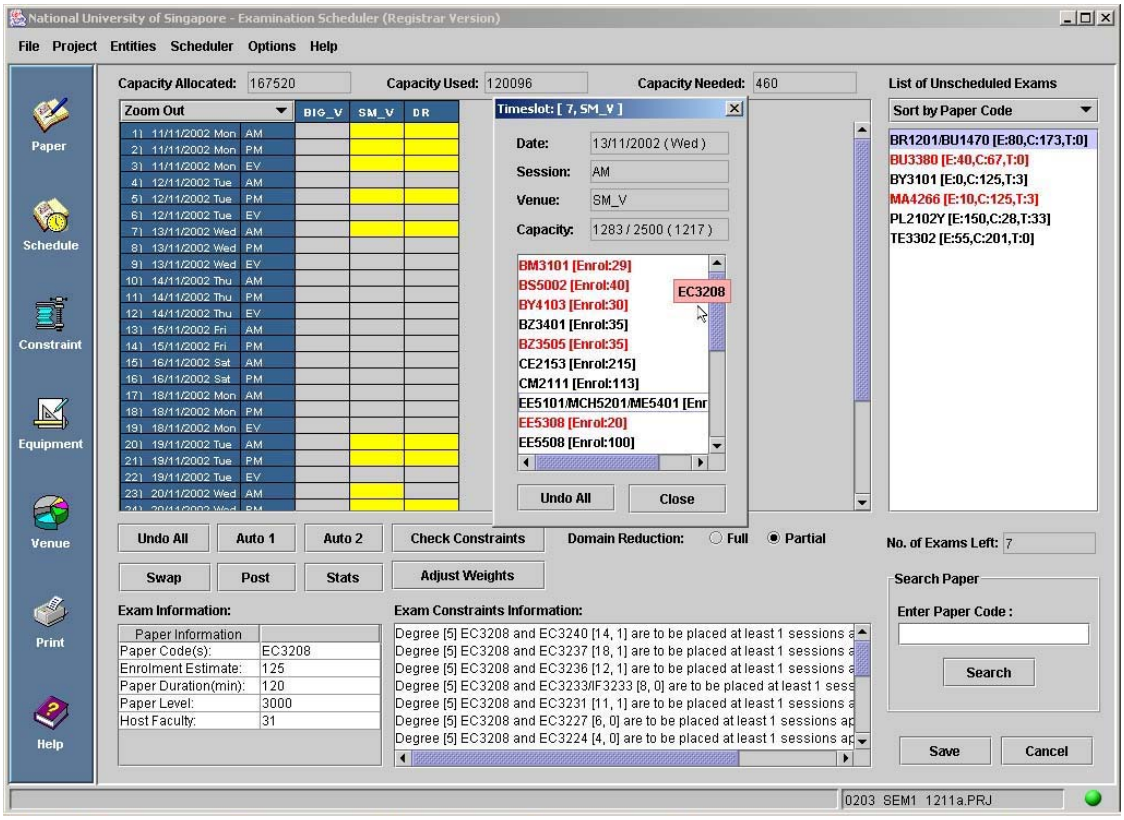

Figure 2: Exam Scheduler

Figure 2 is the screen that allows examination timetable to be scheduled based on enrolment estimates and other timetable constraints entered. The main scheduling window is a cross-table with each row representing an examination session, and each column representing a venue partition (e.g. big venues *BIG_V* and small venues *SM_V*). This window has two views, *Zoom Out* and *Zoom In* view. The *Zoom Out* view allows entire timetable structure to be viewed without giving too many details, and is useful when the user just wish to have an overview of the timetable. The *Zoom In* view expands the cross-table to give the number of examinations and the remaining capacity (in terms of *number of seats filled* / *total capacity*) for each partition/session pair. The timetable may be changed manually. All slots which are available and will not cause any constraints violation for the selected examination will be highlighted to the user.

3.2 Venue

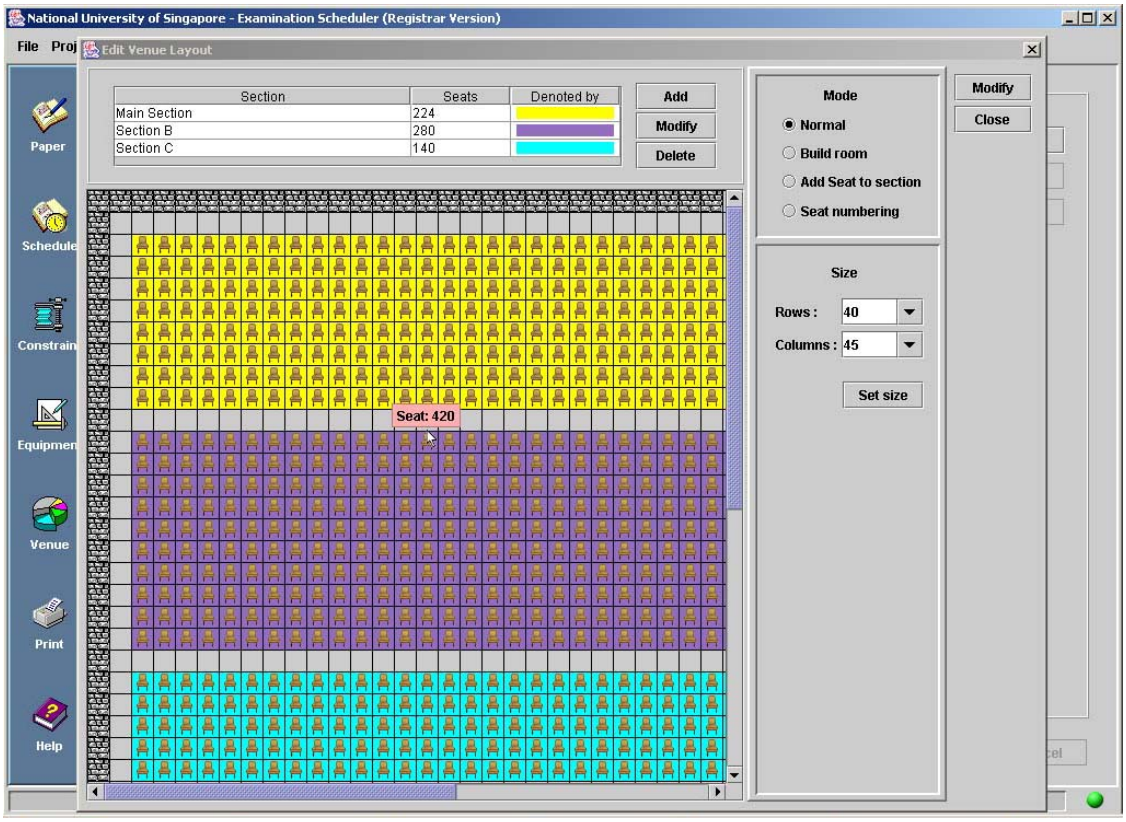

Figure 3: Edit Venue Layout

One of the tasks of an examination timetabling scheduling system is to place examinations into the various venues in various timeslot, subjected to several constraints and to assign each candidate a seat for each examination. As such, particulars like capacity, layout, and available dates of the venues need to be defined. Figure 3 shows one of the screens for editing layout of venues. In this screen, arrangements of walls, doors and seats within a venue can be specified. In order for venues with large number of seats to be more manageable, the facility enables seats to be divided into different sections.

3.3 Venue Assignment

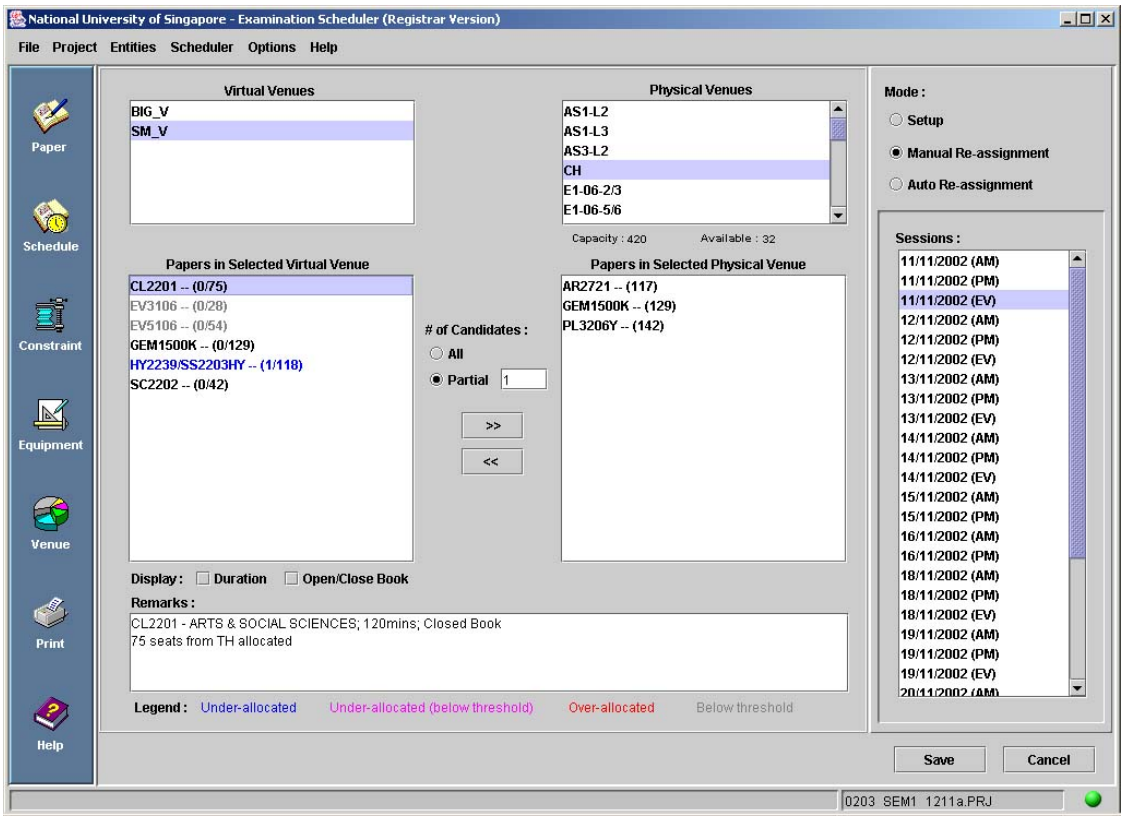

Figure 4: Venue Assignment

After student registration, when actual student data is known for each module, there will be sufficient information to perform venue assignment for each examination. Venues are assigned to examinations such that the number of students that need to travel between venues for consecutive examinations is minimized and that as few examinations as possible have to be held across multiple venues. The venue assignment generated by the engine may be manually altered using the screen shown in Figure 4. Examinations to be held in each venue for every timeslot each day may be viewed in this screen. For each examination, the number of seats allocated to specific venue is also shown. As students sometimes dropped modules half way through the

course, the number of seats allocated may become over-allocated. Problems like this will be highlighted to users.

3.4 Seat Assignment

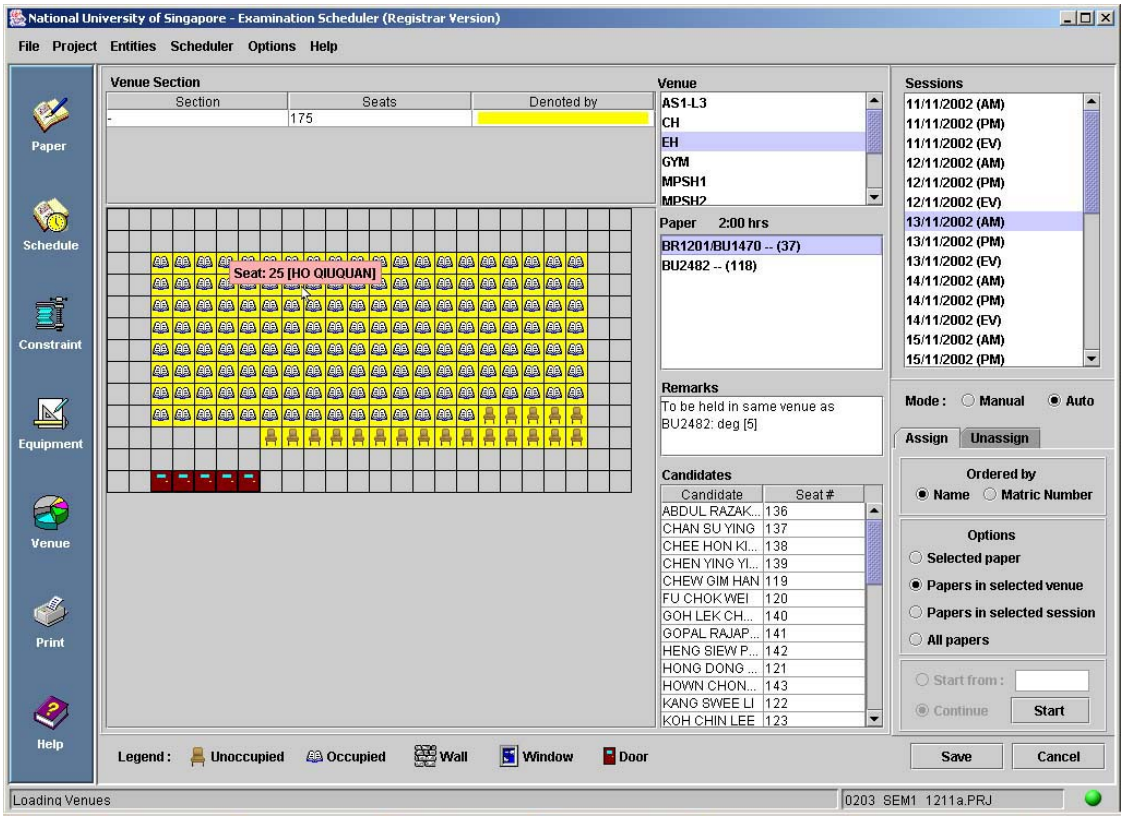

Figure 5: Seat Assignment

Only after examinations have been assigned specific venues, seat assignment can then be performed to assign each candidate a seat for each examination. Figure 5 allows user to either manually or automatically assign seats to each candidate for each examination. The automatic seat assignment engine allocates candidates to their seats in ascending seat number sequence. Candidates will be sorted by either their names or their matriculation numbers, depending on user's selection.

Chapter 4

Heuristics

In this chapter, we discuss a few well-known heuristics, which we used on the venue assignment problem. These heuristics include a greedy method (Greedy), Tabu Search (TS) [Glover 1986, Glover and Laguna 1993], Simulated Annealing (SA) [Kirkpatrick 1983] and "Squeaky Wheel Optimization" (SWO) [Joslin & Clements 1998] [Joslin & Clements 1999].

TS and SA are both forms of *neighbourhood search.* That is, they both involve considering a single solution and then calculating the *neighbourhood* of that solution and move to one of these neighbours. The method by which a solution's neighbourhood is calculated is implementation specific. It could consider all possible solutions arising from moving an item to different knapsacks, or all possible combinations of swapping two items in the assignment, or even both combined. The two methods differ in the criteria used to select which neighbouring solution to move to. Neighbourhood search are often thought of in terms of the underlying landscape where better solutions have lower altitude. A move to a higher quality solution is therefore referred to as a *downhill move* while a move to a lower quality solution is referred to as an *uphill move*.

SWO is a form of iterative variable ordering method. In every iteration, the variables are re-ordered based on the priorities placed on the variables. These priorities are determined by analyzing the solution generated in the previous iteration.

4.1 Greedy Method

The greedy method requires items to be sorted by weight in non-increasing order. At each step, the next item in the list is assigned to the first available knapsack that does not violate capacity and assignment restriction.

4.2 Tabu Search

Tabu Search is a meta-strategy for guiding local search approaches to overcome local optimality. This meta-heuristic has been reported in the literature during the past decade as providing successful solution approaches for a great variety of problem areas.

Tabu Search is based on neighbourhood search with local-optima avoidance in a deterministic way using adaptive memory. The local procedure is a search that uses *move* to define the neighbourhood of a given solution. Memory is implemented by the recording of previously chosen moves using simple but effective data structures known as tabu list. Moves in the tabu list are forbidden for a certain number of iterations and thus prevents cycling and at the same time promotes a more diversified search of the solution through hill-climbing.

An important aspect of Tabu Search is Tabu Operators, which are used to explore neighbouring solutions, thereby creating a *move*, given a solution. Operators that we used in solving the venue assignment problem include a *relocate* operator, *exchange* operator and *multi-exchange* operator which will be discussed in detail in section 4.4. Feasibility of the solution is ensured after the application of an operator.

In each iteration, the list of tabu operators is polled and the operator that gives the best move is selected, even if the resultant solution quality is lower than the current solution. If the selected move is not in the tabu list, it will be used to update the current solution. This chosen move is then updated in the tabu list and will only be released after a pre-determined number of iterations. At times, a taboo-ed move may be allowed if it results in a solution that is better than the best found in preceding iterations. This is known as the inspiration level criterion. We noted that this overriding of the tabu status

Generate a feasible solution S
Initialise tabu list
$non\_improve := 0;$
repeat
    for each operator, select the best move
    from the set of best move, select the best non-taboo-ed move, b
    update S using b
    add b to tabu list
    if S is inferior to the best solution found **then**
        $non\_improve := non\_improve + 1;$
    else
        set current solution to best solution;
        $non\_improve := 0;$
until $non\_improve \geq \max\_non\_improve$.

Figure 6: Tabu Search

of a move will not lead to cycling thereby leading to an already known solution since such an action will only be allowed if a move leads to a better solution. Otherwise, this better solution would already have been found earlier. The Tabu Search algorithm is described in Figure 6.

4.3 Simulated Annealing

Simulated Annealing (SA), like tabu search, is another meta-heuristic method that has a mechanism to escape from local optimum. SA is an iterative stochastic search method derived from the physical annealing process. The SA process starts with a high temperature and is periodically reduced according to some *cooling schedule* which is part of the algorithm's configuration. As in Tabu Search, operators are used to search neighouring solution. Moves that give rise to a better solution are always accepted while deteriorating moves are only accepted with a certain probability. By allowing moves to inferior solutions under the control of a randomized scheme, the chance of getting stuck in a poor local optimum is reduced. Specifically, if a move from the current solution to an inferior solution results in a change of value Δq, this move will be accepted if

$$\exp(-\Delta q / T) > R$$

where T is the current temperature

Δq is the difference in objective value between two solutions

$R \in [0,1]$ is a random number

When the temperature is initially high, many inferior moves are accepted. This acceptance rate drops and inferior moves are nearly always rejected as the temperature drops.

The most important factor of SA in practical application is the cooling schedule which we represent using the following parameters:

- T_{INIT} , the initial temperature,
- f , a function for lowering the temperature,
- T_{TERM} , the terminating temperature,
- $iter$, the maximum number of iterations,
- $local\_iter$, the number of iterations for each temperature

There are basically two types of schedules: static and dynamic. In static cooling schedules, the parameters are fixed and are not changed during the execution of the algorithm. In dynamic cooling schedules, parameters are adaptively changed during the execution of the algorithm.

In our implementation of SA, the dynamic cooling schedule used is a modification of a commonly used static schedule known as the geometric schedule which originates from the early works of Kirkpatrick.

T_{INIT} is set to a sufficiently high temperature such that most of the moves are accepted in the beginning.

In our implementation of f , we used a simple and frequently used decrement function given by

$$t_{k+1} = \alpha \cdot t_k$$

where k is the k^{th} iteration

α is a positive constant close to 1, typically in the range 0.8 to 0.99

Though in theory the SA process should continue until T_{TERM} , the terminating temperature zero, it is a common practice to terminate the process when the chance of accepting an inferior move becomes negligible. T_{TERM} is thus set at some small value close to zero.

$local\_iter$ is ensured of a minimum value at each temperature and increases as the acceptance rate ar decreases. However, to avoid having to perform an extremely large number of iterations when the acceptance rate gets low, $local\_iter$ is bounded by a constant $local\_iter_{\max}$. The function used to determine $local\_iter$ is given by:

$$local\_iter = local\_iter_{\min} + (ar \times local\_iter_{\max})$$

The choice of a cooling schedule radically affects the final solution quality, with faster cooling schedule reaching a local optimum quickly. Slower schedules performed a more thorough search, resulting in a higher quality solution generally, but taking considerably longer to do so.

The simulated annealing algorithm as in the case of minimization is described in Figure 7.

Generate a feasible solution, s

$k := 0;$

$non\_improve := 0;$

repeat

for $l := 1$ **to** $local\_iter$

Selects a neighourhood of the current solution, S

Save the best move, b

if b arise in better solution or $\exp(-\Delta q / T) > R$ **then**

accept b

else reject b

$k := k + 1;$

if best move, b is accepted **then**

Calculate $local\_iter$

Calculate T

if objective(S) < objective(*bestSolution*) **then**

Set current solution to best solution

else $non\_improve := non\_improve + 1$

else $non\_improve := non\_improve + 1;$

until termination criteria reached.

Figure 7: Simulated Annealing

The termination criteria is reached when either

- the number of non-improving moves reached the maximum number of non-improving specified ;
- T_{TERM} is reached;
- or $iter$ is reached.

4.4 Operators

In this section, we shall describe in details the three operators, namely the *relocate operator*, *exchange operator* and *multi-exchange operator* that we used in Tabu Search and Simulated Annealing.

4.4.1 Relocate Operator

The *relocate operator* relocates an item from a knapsack to another knapsack while ensuring that knapsack capacity is not violated and the destination knapsack is admissible to the item. Figure 8 illustrates an example of an item being relocated to another knapsack.

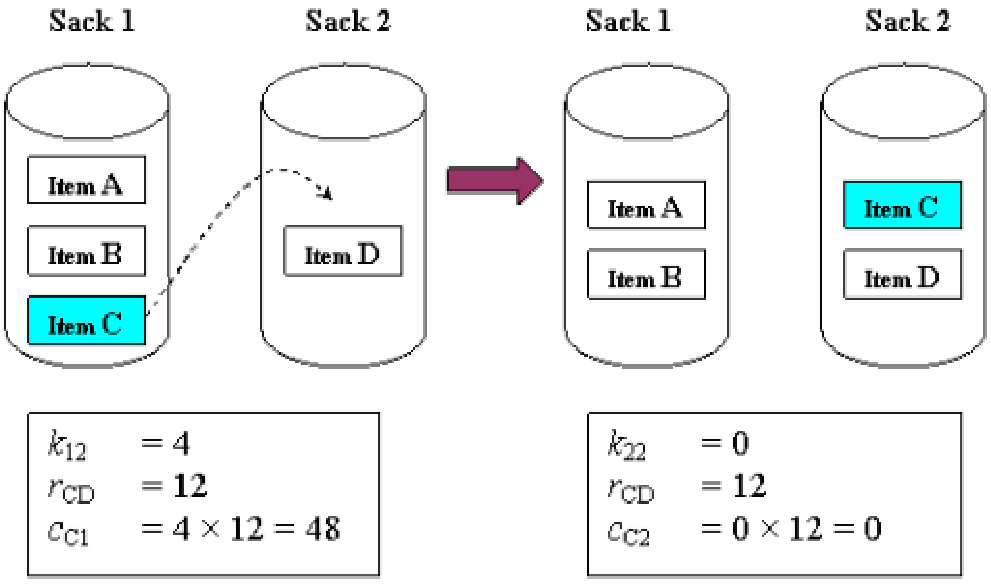

Figure 8: Relocate Operator

When an item is relocated from one knapsack to another, the cost of assigning item to knapsack may change. In the above example, lets consider only the relationship of item C and item D such that the value of r_{CD} is 12. Assumed that the distance measure of knapsack 1 (contains item C) and knapsack 2 (contains item D), k_{12} is 4. Before the relocate operation, the cost of assigning item C is 48. However, after item C is relocated to knapsack 2, the cost of assigning item C becomes 0. This is because item C and D are now in the same knapsack ($k_{22} = 0$). As we can see, relocate operation can potentially minimize the relationship cost. Still, the effectiveness of this operator depends largely on the overall items relationship network as well as the distance measure among the knapsacks. By attempting to reduce the relationship cost of one item with another item may increase its cost with its other related items. Hence, it is important to consider the overall effect in the cost of all related items when an item is relocated from one knapsack to another.

Another advantage of using this operator is that it can potentially relocate an unassigned item to a knapsack whose freed space has earlier been occupied. This helps to achieve the second objective of minimizing the number of unassigned items.

4.4.2 Exchange Operator

In addition to the *relocate operator*, we have the *exchange operator* which exchanges the knapsacks of two items. Similar to the *relocate operator*, the *exchange operator* may help to minimize the overall cost of assigning the two considered items. The *exchange operator* works in one of the two ways.

First: the operator exchanges the knapsacks between two items that already has a knapsack assigned.

Second: the e*xchange operator* may also assign a previously unassigned item to the knapsack in which an item currently resides and in turn, unassigns the latter item from the knapsack if this change results in a better overall solution.

Figure 9 illustrates an example in which the exchange of knapsacks between two items results in a reduction of the overall relationship cost.

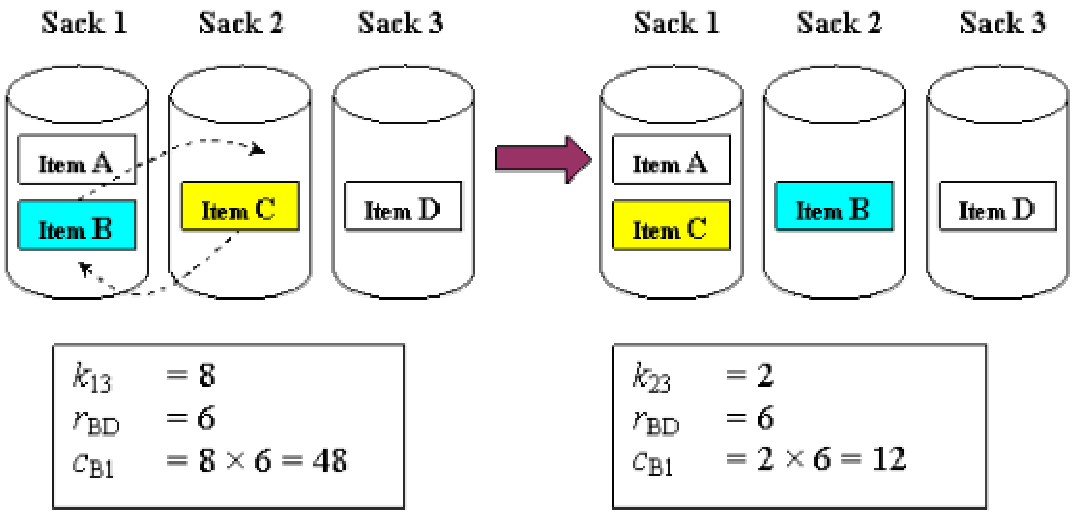

Figure 9: Exchange Operator

In the above example, consider only the relationship of item B and item D such that the value of r_{BD} is 6. Assume that the distance measure of k_{13} and k_{23} is 8 and 2 respectively. Before the exchange operation, the cost of assigning item B is 48. After exchanging knapsacks of item B and C, the cost of assigning item B is reduced to 12. This is due to the fact that item B is shifted to a knapsack that has a lower distance

measure with respect to knapsack 3 (where item D is in). Again, as with the *relocate operator*, the overall cost savings resulted from an *exchange operation* has to take into account of all the related items of the two exchanged items.

One limitation of the *relocate operator* is that it is very difficult to relocate items to knapsacks that are filled almost completely. With the *exchange operator*, it is possible to exchange items between two tightly filled knapsacks, as long as the size of the item in consideration from the first knapsack plus the remaining capacity of its knapsack is greater than the size of the item from the second knapsack, and vice versa. Thus, this operator gives additional flexibility to the number of possible moves the search heuristic can make.

4.4.3 Multi-Exchange Operator

The *relocate* and the *exchange* operators bring only a small change at a time to the solution as at most two items were involved during the operation. It may take a long time for the solution to converge. In addition, the operators are very restrictive due to the presence of capacity and knapsack admissibility constraints. As such, the operators may not be effective to bring about a big change to the overall solution.

The *multi-exchange operator* attempts to rectify this shortcoming by allowing more items to be involved during its operation. The operator selects two knapsacks and unassigns all items that have been assigned to these two knapsacks. These items together with the list of previously unassigned items will be reconsidered for assignment to the two selected knapsacks. A greedy algorithm is then used to assign these items to the two knapsacks.

4.5 “Squeaky Wheel” Optimization

The key idea of SWO is a Construct/Analyze/Prioritize cycle, as illustrated in Figure 10. Elements of the problem are placed in a priority queue. These elements are ordered by their priority values based on some predefined measure. A solution is generated by the constructor using a greedy algorithm, making decisions based on this order. The solution is then analyzed to find the problematic elements of the solution. Problematic elements refer to those elements that contributed to the poor solution. For example, in VAP in which minimizing student movement is one of the objectives, examinations that have student movement are penalized. A blame factor is assigned to these problematic elements, increasing their priorities. Difficult elements with their higher priority value are thus nearer to the front of the sequence and handled sooner by the constructor on the next iteration. As elements at the front of the sequence tends to be better handled, and results in a lower blame factor. This whole process is performed iteratively until a termination condition occurs.

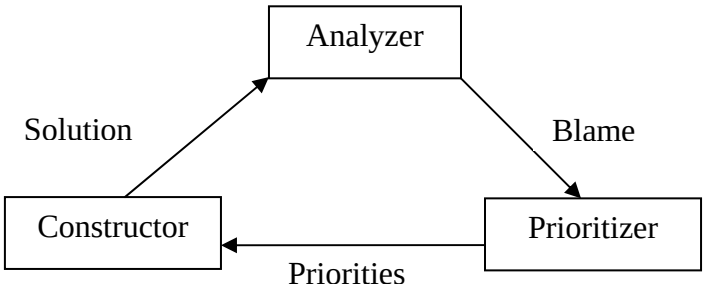

Figure 10: The Construct/Analyze/Prioritize cycle

The three main components of SWO are:

Constructor

Given a sequence of problem elements, the constructor generates a solution using a greedy algorithm. This sequence determines the order in which decision will be made.

Analyzer

The analyzer is responsible for assigning a numeric "blame" factor to elements that contribute to defects in the current solution. By analyzing the solution, difficult elements are differentiated from the easier ones and be given the appropriate attention in the next iteration. Though this information about the problem structure is local and may only apply to the search space currently under examination, it is useful in determining which direction the search should head towards next.

Prioritizer

The prioritizer uses the blame factor assigned by the analyzer to modify the previous sequence of the problem elements. Elements that are blamed are moved towards the front of the priority queue. The higher the blame, the further the element will be moved.

Priority sequence of items plays an important role in SWO. Depending on the amount of blame placed on the elements, elements get shifted up and down the priority queue. As a difficult element move forward in the sequence, it will be handled sooner by the constructor and tends to be handled better, its blame factor decreases as well. Difficult to handle elements thus move rapidly up the sequence to a position they will

be handled well. Once that happens, the blame factor assigned to them drops, causing them to slowly slide down the sequence as other part of the problem that has not been handled well are given more priority. As the algorithm iterates, elements that are always easy to handle sink to the back of the sequence and stay there, while more difficult elements move back and forth the priority queue trying to get into a position they can be handled well without depriving others.

SWO was chosen as one of our heuristics because of its suitability to our problem. In MKPIR, some elements may be more difficult to handle than others. For example, some items can be assigned to only a few knapsacks, while others allow for much more flexibility. Then some items have many relationships with other items while others have none at all. Clearly, changes in the sequence of items to which items are assigned knapsack make a difference to the overall assignment. Even a small change to the sequence can result in great consequences for any items that are behind it, since items with lower-priority can only be assigned to the leftover knapsack after higher-priority items have been assigned. As the difficulty of the assignment of an item is reflected by its priority value, these changes will be beneficial to the overall solution.

Implementation

At the constructor, items are assigned to the knapsack that will not result in any *relationship cost* or one with minimum *relationship cost* if that is inevitable, subjected to *knapsack admissibility constraint* and knapsack capacity constraint.

At the analyzer, a blame factor is assigned to:

- items that are not assigned any knapsack ;
- items that results in *relationship cost*;

The numeric blame value assigned to an item for the different type of blame is as follows:

Item not assigned any knapsack

A simple method will be to assign size of the unassigned item as the blame value. However, this method will not be desirable if size of the items differs greatly. Items which are small in size but restrictive in the knapsacks it can be assigned to, will never make it to the front of the priority queue. On the other hand, a large item if not assigned, may quickly jump right to the front of the queue and never move down again. We mentioned earlier that once an item is handled well, its blame factor will decrease and it will slowly slide down the queue when the blame factor of other mishandled items increase. However, if size of the item is used as the blame value, it will take many iterations before the blame factor of a small and difficult item surpassed that of the large item.

Therefore, to prevent large item from domineering the front of the queue, a function is required so that small items get their fair chance of climbing up the priority queue.

In our implementation, the penalty value inflicted on items that are not assigned to any knapsack is computed as follows:

$$\textit{penalty value} = \begin{cases} \frac{1}{10} \times \textit{ItemSize} & \text{if } \textit{ItemSize} \geq 1000 \\ \frac{1}{5} \times \textit{ItemSize} & \text{if } 500 \leq \textit{ItemSize} < 1000 \\ \frac{1}{4} \times \textit{ItemSize} & \text{if } 300 \leq \textit{ItemSize} < 500 \\ \frac{1}{2} \times \textit{ItemSize} & \text{if } 100 \leq \textit{ItemSize} < 300 \\ \textit{ItemSize} & \text{if } 50 \leq \textit{ItemSize} < 100 \\ 10 \times \textit{ItemSize} & \text{if } \textit{ItemSize} < 50 \end{cases}$$

where *ItemSize* is the size of the item.

Item that results in relationship cost

The penalty value assigned to items that result in relationship cost is the total cost of its relationship with other related items.

4.6 Combining "Squeaky Wheel" Optimization and Tabu Search

In SWO, moves in the solution space are made indirectly, via the re-prioritization of elements that result from analyzing the previous solution. In each iteration, a solution is created from scratch based on this prioritization of the elements. A small change in the sequence of the elements may thus correspond to a large change

in the corresponding solution generated by the constructor, compared to the solution from the previous iteration. Though the ability of SWO to make a single large move in the solution space is a strength of the approach, it is also a weakness. SWO is poor at making small “tuning” moves in the solution space. As suggested by [Joslin & Clements 1999], SWO could be combined with local search to look for improvements in the vicinity of good solution. Incorporating an effective local search in the constructor may however considerably slow down the speed of SWO. Therefore, we have attempted to apply Tabu Search to the resultant solution of SWO only after SWO terminates.

Chapter 5

Experimental Results

5.1 Experimental Data

In order to evaluate the effectiveness of our proposed heuristics, sets of real enrolment data together with the examination timetable schedule produced by the examination timetable scheduler in *UTTSExam* was used for testing. In addition, problem instances with different characteristics were randomly generated. We would like to see how these different parameterizations affect the performance of the heuristics.

5.1.1 Actual Data

We obtained student registration data from the NUS Computer Centre for semester 2 of the academic year 2001/2002 (**0102_sem2)**, semester 1(**0203_sem1**) and semester 2 (**0203_sem2**) for academic year 2002/2003, venue constraints for each examinations and the examination timetable schedule without actual venues from *UTTSExam.* The data consisted of a set of text files containing the list of student-examination tuples and the examination timeslot assigned to each examination. Table 1 to Table 3 shows the characteristics on the set of data for each of the semesters. Examinations are held over a period of 12 days. *Number of oversize examinations* refers to the number of examination whose number of candidates is more than the capacity of any of the available venues. It is therefore not possible to assign a venue to

Day	Number of Examinations	Number of Over Size Examinations	Number of Relations	Average Relation Size
1	111	1	96	6
2	80	0	108	7
3	100	0	102	3
4	80	0	108	3
5	84	0	79	2
6	114	0	166	1
7	105	0	160	2
8	88	0	124	2
9	90	0	87	3
10	77	0	57	3
11	61	0	29	2
12	59	0	18	5

Table 1: Characteristics of **0102_sem2**

Day	Number of Examinations	Number of Over Size Examinations	Number of Relations	Average Relation Size
1	48	1	78	8
2	41	1	52	8
3	44	0	85	2
4	48	0	113	6
5	36	1	29	5
6	36	0	18	8
7	48	0	79	6
8	48	0	124	3
9	46	0	108	3
10	42	0	101	3
11	27	1	9	1
12	31	0	16	4

Table 2: Characteristics of **0203_sem1**

Day	Number of Examinations	Number of Over Size Examinations	Number of Relations	Average Relation Size
1	89	2	89	8
2	95	0	137	2
3	93	3	172	3
4	112	0	165	2
5	83	1	64	3
6	69	1	70	2
7	108	0	162	2
8	105	0	142	2
9	95	0	139	2
10	73	0	63	2
11	69	1	57	1
12	76	1	58	2

Table 3: Characteristics of **0203_sem2**

examination of this nature unless the examination is held in multiple venues. Number of relations refers to the number of back-to-back examinations.

Table 4 gives the list of examination venues that are used for scheduling examinations for academic year 2001/2002, semester 2. The Suntec City Exhibition Hall is an off-campus venue rented by NUS for examination purpose, while the rest of the venues are on-campus venues. On-campus venues are relatively closer to each other. We have assigned a value of one unit to denote the distance between Suntec City Exhibition Hall and the rest of the on-campus venue. A unit of zero is assigned to denote the distance between the on-campus venues. For most of the examinations, all the venues available in that session are admissible except the two drawing rooms. Usage of the two drawing rooms is restricted to examinations which require large drawing papers.

Venue	**Capacity**
Suntec City Exhibition Hall	1600
Gymnasium	312
Multi-Purpose Sports Hall 1	750
Multi-Purpose Sports Hall 2	850
Competition Hall	396
Eusoff Hall	175
Lecture Theatre 8	117
Lecture Theatre 11	125
Lecture Theatre 13	81
Lecture Theatre 17	112
Drawing Room 1	62
Drawing Room 2	54

Table 4: List of Examination Venues for **0102_sem2**

From academic year 2002/2003, off-campus commercial venues are no longer rented to hold examinations. Big and small venues suitable for holding examinations from all over the campus are used instead. We have assigned a value of one unit to

denote the distance between venues that are not within walking distance and a unit of zero to others that are within a five minutes walk. As most of the smaller venues are owned and managed by the faculties, these venues are restricted to only examinations conducted by the faculty. Table 5 and Table 6 shows the examination venues that can be used for academic year 2002/2003 semester 1 and semester 2 respectively.

Venue	Capacity	Allowed Users
Multi-Purpose Sports Hall 1	800	All
Multi-Purpose Sports Hall 2	900	All
Multi-Purpose Sports Hall 4	90	All
Gymnasium	312	All
Competition Hall	420	All
Eusoff Hall Function Room	175	All
Temasek Hall Multi-Purpose Room	111	All
Pgp Residences Multi-Purpose Hall	220	All
Engrg Blk 1, #06-02/03	72	Engineering
Engrg Blk 1, #06-05/06	65	Engineering
Engrg Blk 1, #06-07/08	65	Engineering
Engrg Blk 1, #06-09/10	70	Engineering
Engrg Blk 3, #06-06/07	65	Engineering
Engrg Blk 3, #06-08/09	65	Engineering
Engrg Blk 3, #06-10/11	70	Engineering
Engrg Blk 3, #06-02/03	72	Engineering
Engrg Blk 5, #03-04/05	75	Engineering
Engrg Blk 5, #03-06/07	72	Engineering
Faculty of Arts & Social Sciences Blk 1, #02-01/04	120	Arts & Social Sci
Faculty of Arts & Social Sciences Blk 1, #03-01/04	130	Arts & Social Sci
Faculty of Arts & Social Sciences Blk 3, #02-12/15	90	Arts & Social Sci
Faculty of Arts & Social Sciences Blk 6, #02-12/15	160	Arts & Social Sci
Lecture Theatre 8	117	Arts & Social Sci
Lecture Theatre 9	74	Arts & Social Sci
Lecture Theatre 11	125	Arts & Social Sci
Lecture Theatre 13	81	Arts & Social Sci
Tutorial Wing, Level 3, S4A Building	150	Science
Science Blk 13, Level 5	110	Science
Tutorial Wing, Room 30 To 41, Level 4, S16 Building	110	Sch. Of Computing

Table 5: List of Examination Venues for 0203_sem1

For each venue, the faculties that may conduct examinations in that venue is shown. Noticed that although the venues available in semester 2 2002/2003 may be

Venue	Capacity	Allowed Users
Multi-Purpose Sports Hall 1	572	All
Multi-Purpose Sports Hall 2	660	All
Multi-Purpose Sports Hall 4	60	All
Gymnasium	230	All
Competition Hall	348	All
Eusoff Hall Function Room	136	All
PGP Residences Multi-Purpose Hall	168	All
Engrg Blk 1, #06-02/03	72	Engineering
Engrg Blk 1, #06-05/06	65	Engineering
Engrg Blk 1, #06-07/08	65	Engineering
Engrg Blk 1, #06-09/10	70	Engineering
Engrg Blk 3, #06-06/07	65	Engineering
Engrg Blk 3, #06-08/09	65	Engineering
Engrg Blk 3, #06-10/11	70	Engineering
Engrg Blk 3, #06-02/03	72	Engineering
Engrg Blk 5, #03-04/05	75	Engineering
Engrg Blk 5, #03-06/07	72	Engineering
Lecture Theatre 7A	100	Engineering
Engineering Auditorium	100	Engineering
Faculty of Arts & Social Sciences Blk 1, #02-01/04	81	Arts & Social Sci
Faculty of Arts & Social Sciences Blk 1, #03-01/04	75	Arts & Social Sci
Faculty of Arts & Social Sciences Blk 3, #02-12/15	55	Arts & Social Sci
Faculty of Arts & Social Sciences Blk 6, #02-09/14	102	Arts & Social Sci
Lecture Theatre 8	117	Arts & Social Sci
Lecture Theatre 9	74	Arts & Social Sci
Lecture Theatre 11	125	Arts & Social Sci
Lecture Theatre 13	81	Arts & Social Sci
Tutorial Wing, Level 3, S4A Building	79	Science
Science Blk 13, Level 5	110	Science
Science 1A, #02-12	40	Science
Science Blk 4A, Level 1 & 2	88	Science
S4A, Level 3 Tutorial Wing Rooms 6 To 10	56	Science
Science Library	350	Science
Tutorial Wing, Room 30 To 41, Level 4, S16 Building	110	Sch. of Computing
S16, Level 3 Seminar Room 1	65	Sch. of Computing
Lecture Theatre 27	150	Sch. of Computing
SDE2 #03-15/16, Executive Classrooms 4 & 5	60	SDE
SDE2 #03-12/13, E Studio	60	SDE
Lecture Theatre 16	95	Sch. of Business
Lecture Theatre 17	100	Sch. of Business
Hon Sui Sen Auditorium	50	Sch. of Business
Moot Court, Faculty Of Law	64	Law
Faculty Conference Rm, Faculty Of Law	45	Law
Blk Admin Level 7 Conf Rom	64	USP
Blk Admin, Level 7 Tr3	64	USP
Blk Admin, Level 7 Tr5	64	USP
Blk Admin, Level 5 Sr4	64	USP
Blk Admin, Level 5 Sr5	64	USP
USP Blk Adm	70	USP

Table 6: List of Examination Venues for 0203_sem2

similar to that that is available in semester 1 2002/2003, the capacity available for each venue is about 20% to 30% less than the previous semester. This reduction was due to the measures taken by the University's during the Severe Acute Respiratory Syndrome outbreak. To reduce close contact between candidates, candidates are seated at least one metre apart in the examination halls.

5.1.2 Generated Problem Instances

In addition to the actual data we have obtained from NUS Computer Centre, three types of randomly generated problem instances are also considered. For each type, admissibility density of knapsacks A_i ranging from 0.1 to 1.0 is tested. The problem instances are created such that for the first problem type, the number of relationship between items is high. The other two types are designed such that the N/M ratio is large. For the second problem type, several items are to be fitted into a few knapsacks, while the last type has many knapsacks and a few items.

For the **many relationships** instances, w_i is randomly distributed in the interval $[10,100]$, v_y is randomly distributed in the interval $[10,200]$ and the number of r_{ij} for each item is randomly distributed in the interval $[10,40]$.

For the **many items and few knapsacks** instances, w_i is randomly distributed in the interval $[1,150]$, v_y is randomly distributed in the interval $[1000,5000]$ and the number of r_{ij} for each item is randomly distributed in the interval $[0,10]$.

For **the few items and many knapsacks** instances, w_i is randomly distributed in the interval $[50,300]$, v_y is randomly distributed in the interval $[50,300]$ and the number of r_{ij} for each item is randomly distributed in the interval $[0,10]$.

For all instances, the value of r_{ij} is randomly generated such that $w_i \le r_{ij} \le w_j$.

Table 7 shows the characteristics of each of the problem instances generated based on the above parameter settings.

Problem Instance	Many Relationship Instance	Many Items, Few Sacks Instance	Few Items, Many Sacks Instance
Number of Items	60	1000	60
Number of Sacks	10	10	40
Total Item Size	1223	23646	4772
Total Sack Capacity	1499	30208	6800
Total Number of Relations	853	3851	228
Average Number of Relations	14	3	3
Average Size of a Relation	6	4	30

Table 7: Generated Problem Instances

5.2 Evaluation of the Assignment Quality

Though we have made the assumption that the total size of all N items is less than or equals the total capacity of the M knapsacks in MKPIR, not all items may be assigned a knapsack even in the optimal solution. Due to assignment restrictions, not all knapsacks are admissible to an item. As a result, it is not always possible to assign all items a knapsack even if $\sum_{i=1}^{N} w_i \le \sum_{y=1}^{M} v_y$.

Consider a problem instance which consists of four items, $w_1 = 40$, $w_2 = 30$, $w_3 = 20$, $w_4 = 10$ with assignment restriction $A_1 = \{v_1\}$,

$A_2 = \{v_1, v_2\}$, $A_3 = \{v_2, v_3\}$, $A_4 = \{v_3\}$ and three knapsacks, $v_1 = 50$, $v_2 = 40$, $v_3 = 20$. Total size of the items to pack is 100 and the packing potential of the knapsacks is 110. An optimal solution would be to assign w_1 to v_1, w_2 to v_2, w_3 to v_3 while w_4 is left unassigned. Clearly, there exists no solution that is able to assign all items a knapsack for the problem instance stated.

In the venue assignment problem that motivated our study of the MKPIR problem, the objective is to minimize total relationship cost incurred while maximizing assigned weight. With the two objectives, the choice of the knapsack to which an item is assigned becomes more critical. A solution that minimizes cost relationship incurred does not necessarily maximize assigned weight.

One popular method of assessing the quality of solution to optimization problems with multiple objectives is the weighted sum strategy. The weighted sum strategy converts the multi-objective problem into a single-objective problem by constructing a weighted sum of all the objectives. A weighting coefficient has to be attached to each of the objectives.

$$F(x) = \sum_{i=1}^{k} d_i \cdot f_i(x), \quad \text{where} \sum_{i=1}^{k} d_i = 1$$

The problem is, for the weights to reflect closely the importance of the objectives; all functions have to be expressed in units of approximately the same numerical values. Normalization of the functions is thus required since the measures used for relationship cost and weights can be very different. In addition, by analyzing one single value, it is difficult to tell which objective is doing better. For example, given a 2-objective function $F(Obj1, Obj2) = d_1 \times Obj1 + d_2 \times Obj2$ where d_1 and d_2

are weight constants and $d_1 + d_2 = 1$. Assuming $d_1 = 0.8$ and $d_2 = 0.2$, if one solution returns $F(8,4)$ and another solution returns, $F(7,8)$ both of them will have the same objective value of 7.2. In this case, both solutions will be considered equal, which makes it difficult to distinguish the goodness of one over the other. Furthermore, by varying the weights, we can arrive at different best solution for the same problem using the same heuristic. This makes the setting of weights' value crucial and ambiguous.

After considering the above, we adopt a prioritized objective function which places the priority of one objective over the others. For the venue assignment problem, the priority of our three objectives is ranked as follows:

1. minimize total size of unassigned examinations
2. minimize total number of unassigned examinations
3. minimize student movement, $\sum_{x=1}^{M}\sum_{i=1}^{N} c_{ix} z_{ix}$

One advantage of this measure is that the comparison of goodness of two solutions becomes unambiguous. As in the earlier example, if *Obj1* has a higher priority over *Obj2*, then the solution having $F(8,4)$ is clearly better than the solution $F(7,8)$. This measure of solution quality is commonly adopted in other multi-objective optimization problems such as the vehicle routing problem where the objective "number of vehicles used" is normally deemed to have a higher priority over the objective "total distance traveled by all vehicles"

5.3 Implementation Matters

In chapter 4, we presented a few algorithms, namely a greedy algorithm, the Tabu Search Algorithm, the Simulated Annealing Algorithm and the Squeaky Wheel Optimization Algorithm. These algorithms were coded in JDK1.2 using *IBM VisualAge^{TM} for Java*. All experiments are performed on a Pentium IV 2.4GHz PC with 512MB RAM and computational times are rounded to the nearest second.

In the Tabu Search Algorithm, size of the tabu list is set to 20. The algorithm terminates if it cannot find a move that improves the best found solution after 1000 consecutive iterations.

In the Simulated Annealing algorithm, the initial and terminating temperature is set to 400 and 0.001 respectively. Cooling rate for the annealing process is set to 0.988. For each problem instance, SA runs a maximum of 100000 iterations. SA terminates if it is not able to find a move that improves the global best solution after 1000 consecutive iterations. At each temperature, the minimum number of local iterations is set to be 20 and the maximum number of local iterations allowed is 250.

In SWO, the algorithm is allowed to run for 10000 iterations before it is terminated.

In our hybrid approach, in which we applied Tabu Search on the solution obtained from SWO, SWO is also allowed to run for 10000 iterations before it is terminated. Size of the tabu list for Tabu Search is set to be 20 while the maximum number of non-improving moves is set to be 500.

All the heuristics are set with the primary task of assigning all items a knapsack given the available capacity and the secondary task of minimizing the relationship cost between the assigned items.

5.4 Results

The breakdown of results is given as the size of items not assigned, number of items not assigned and the relationship cost incurred. Relationship cost is calculated only for items that have been assigned knapsacks. Items that are not assigned knapsacks do not incur relationship cost since the relationship cost of an item largely depends on the knapsack it is assigned to.

Table 8 to Table 10 show the results achieved for the actual data using the heuristics. The results achieved are also shown graphically in Figure 11 to Figure 13.

Day	Greedy			Tabu Search			SA			SWO			SWO + Tabu Search		
	Size not assign	#not assign	Relation cost	Size not assign	#not assign	Relation cost	Size not assign	#not assign	Relation cost	Size not assign	#not assign	Relation cost	Size not assign	#not assign	Relation cost
1	1634	1	143	1634	1	96	1634	1	45	1634	1	36	1634	1	25
2	328	2	370	328	2	313	328	2	307	328	2	229	290	2	135
3	0	0	141	0	0	63	0	0	61	0	0	38	0	0	30
4	0	0	133	0	0	47	0	0	64	0	0	30	0	1	30
5	0	0	72	0	0	62	0	0	45	0	0	36	0	0	31
6	0	0	161	0	0	49	0	0	48	0	0	37	0	0	36
7	0	0	248	0	0	174	0	0	110	0	0	65	0	0	58
8	0	0	177	0	0	103	0	0	133	0	0	27	0	0	19
9	0	0	106	0	0	69	0	0	34	0	0	28	0	0	26
10	0	0	142	0	0	46	0	0	44	0	0	39	0	0	26
11	0	0	54	0	0	23	0	0	23	0	0	31	0	0	7
12	0	0	3	0	0	0	0	0	0	0	0	1	0	0	0

Table 8: Results for **0102_sem2**

On the **0102_sem2** problem, due to the availability of a huge venue with capacity 1600, all the heuristics are able to assign a venue to each of the examinations on most of the days. The best result is obtained by using the hybrid approach which gives the lowest relationship cost for all of the days. Generally, SWO outperforms Tabu Search and SA, while the results for Tabu Search and SA are comparable. Similarly for the **0203_sem1** problem, the best result is found using the hybrid approach for consistently all the days. On the **0203_sem2** problem, hybrid approach generally did better than the rest of the heuristics except on the second and the third day when SA did better. For both the **0203** data sets, the overall results found by Tabu Search, SA and SWO are comparable. The difference in results in terms of the size and number of examinations

Day	Greedy			Tabu Search			SA			SWO			SWO + Tabu Search		
	Size not assign	#not assign	Relation cost	Size not assign	#not assign	Relation cost	Size not assign	#not assign	Relation cost	Size not assign	#not assign	Relation cost	Size not assign	#not assign	Relation cost
1	1208	3	28	1208	3	2	1208	3	7	1208	3	3	1208	3	2
2	1286	1	12	1286	1	8	1286	1	5	1286	1	1	1286	1	1
3	0	0	41	0	0	2	0	0	1	0	0	1	0	0	1
4	115	1	150	115	1	90	115	1	92	115	1	106	115	1	67
5	961	1	18	961	1	0	961	1	0	961	1	0	961	1	0
6	715	4	38	715	4	38	715	4	38	715	4	31	638	3	37
7	0	0	40	0	0	13	0	0	9	0	0	11	0	0	8
8	0	0	124	0	0	49	0	0	14	0	0	2	0	0	2
9	141	1	52	0	0	2	0	0	3	0	0	2	0	0	2
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	1549	1	0	1549	1	0	1549	1	0	1549	1	0	1549	1	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 9: Results for **0203_sem1**

Day	Greedy			Tabu Search			SA			SWO			SWO + Tabu Search		
	Size not assign	#not assign	Relation cost	Size not assign	#not assign	Relation cost	Size not assign	#not assign	Relation cost	Size not assign	#not assign	Relation cost	Size not assign	#not assign	Relation cost
1	2022	3	77	2022	3	66	2022	3	53	2022	3	49	2022	3	49
2	390	4	202	282	3	152	390	4	163	390	4	143	250	2	142
3	2681	4	121	2681	4	101	2681	4	79	2681	4	113	2681	4	93
4	0	0	173	0	0	117	0	0	118	0	0	150	0	0	122
5	1469	1	39	1469	1	34	1469	1	29	1469	1	24	1469	1	22
6	1191	4	77	1191	4	61	887	2	67	1191	4	62	887	2	60
7	237	3	219	154	2	171	125	1	157	237	3	215	125	1	164
8	0	0	102	0	0	48	0	0	44	0	0	23	0	0	21
9	0	0	87	0	0	72	0	0	68	0	0	68	0	0	61
10	102	1	65	102	1	59	102	1	63	102	1	48	0	0	49
11	1532	1	18	1532	1	7	1532	1	0	1532	1	2	1532	1	1
12	729	1	0	729	1	0	729	1	0	729	1	0	729	1	0

Table 10: Results for **0203_sem2**

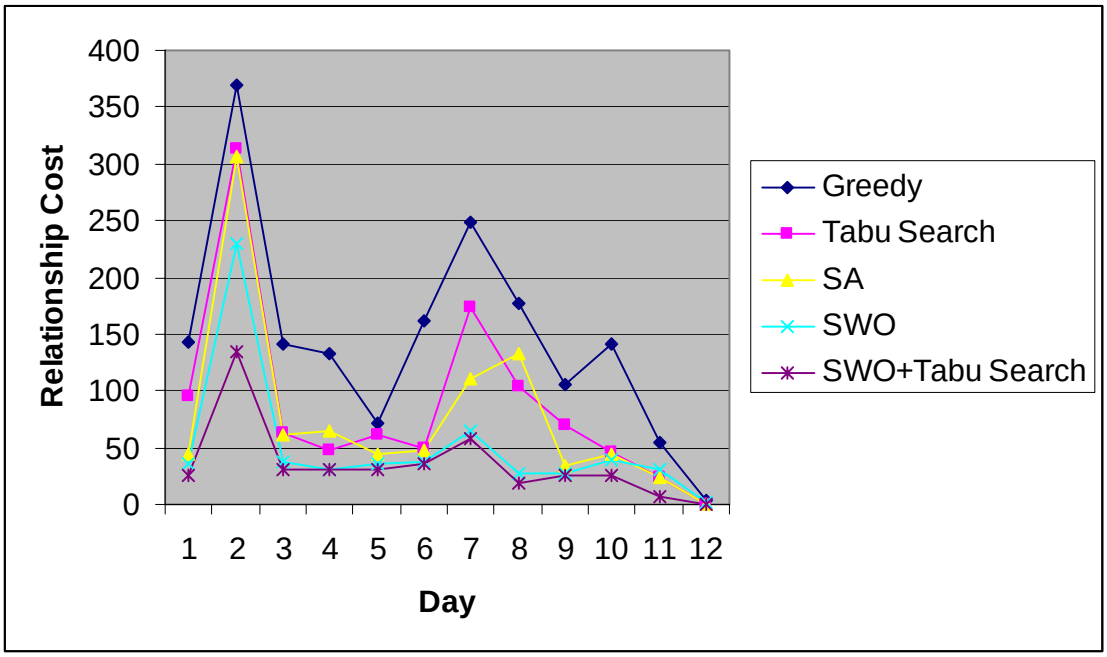

Figure 11: Relationship Cost for **0102_sem2**

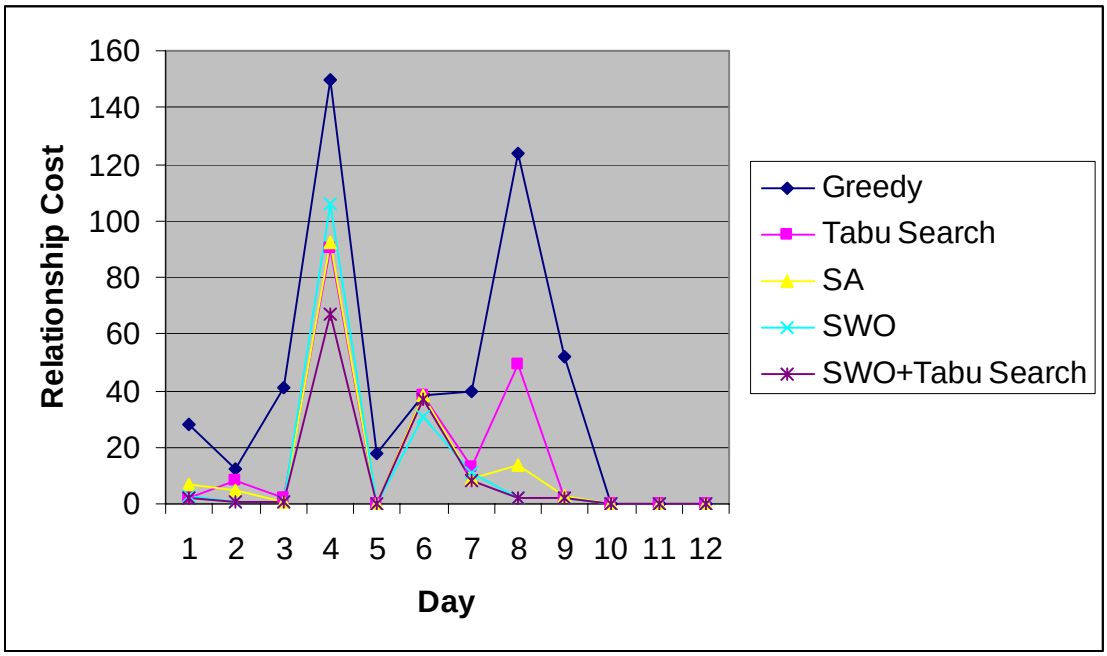

Figure 12: Relationship Cost for **0203_sem1**

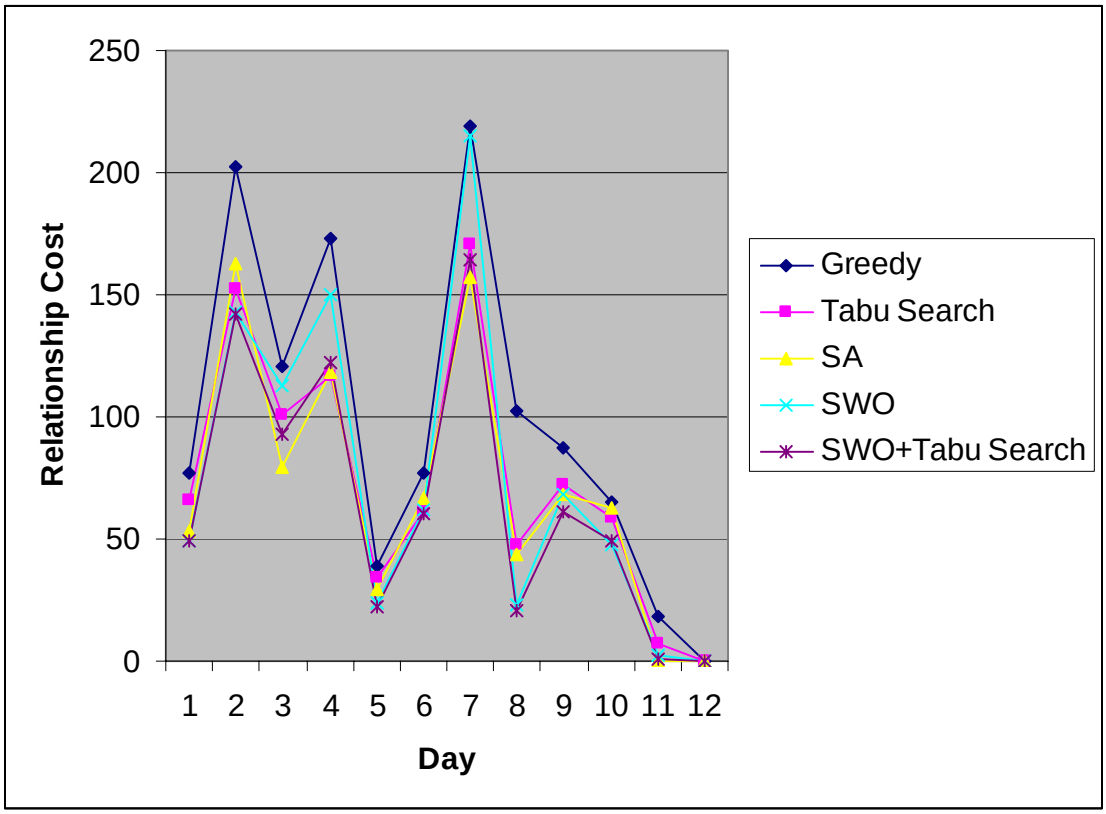

Figure 13: Relationship Cost for **0203_sem2**

that could be held in one single venue is not great for all three actual data sets. As the ratio between the capacity made available and capacity required is rather high, this makes it relatively easy for the heuristics to fit in most of the average size examinations. An examination whose size is greater than the largest available venues remains unassigned since these examinations are only assigned to multiple venues after the termination of our heuristics and is not reflected in the results.

Computational time taken by the various heuristics for each of the actual data sets is reflected in Table 11 to Table 13. For all three data sets, though the greedy method requires the least amount of time, it clearly performs badly when its results is compared with that that is found by the other heuristics.

Day	Greedy	Tabu Search	SA	SWO	SWO + Tabu Search
1	1	51	61	13	23
2	1	34	29	10	19
3	1	38	37	14	22
4	1	36	29	12	20
5	1	31	27	10	17
6	1	55	39	20	31
7	1	52	43	17	27
8	1	45	47	14	22
9	1	33	32	11	19
10	1	25	19	8	17
11	1	17	19	4	10
12	1	15	15	4	9

Table 11: Computational Time for **0102_sem2**

Day	Greedy	Tabu Search	SA	SWO	SWO + Tabu Search
1	1	141	74	6	24
2	1	68	29	4	16
3	1	97	40	7	18
4	1	281	92	7	23
5	1	21	14	3	8
6	1	24	6	2	9
7	1	91	40	7	19
8	1	122	82	10	22
9	1	112	48	10	27
10	1	104	18	10	22
11	1	15	4	2	7
12	1	17	5	3	7

Table 12: Computational Time for **0203_sem1**

Day	Greedy	Tabu Search	SA	SWO	SWO + Tabu Search
1	1	336	104	11	74
2	1	549	134	12	70
3	1	484	196	15	76
4	1	808	163	19	72
5	1	121	57	8	30
6	1	141	64	7	31
7	1	607	274	15	69
8	1	490	232	16	82
9	1	510	232	15	58
10	1	148	50	8	28
11	1	129	45	10	30
12	1	291	27	11	54

Table 13: Computational Time for **0203_sem2**

Density	Greedy			Tabu Search			SA			SWO			SWO + Tabu Search		
	#not assign	Size not assign	Relation cost	#not assign	Size not assign	Relation cost	#not assign	Size not assign	Relation cost	#not assign	Size not assign	Relation cost	#not assign	Size not assign	Relation cost
0.1	224	14	13566	198	12	15005	198	12	15005	220	14	12043	84	2	20346
0.2	62	5	20978	26	2	22217	10	1	23060	50	4	16661	0	0	19890
0.3	70	6	20599	16	1	19584	70	6	20599	11	1	22976	0	0	19281
0.4	55	5	19903	0	0	18624	0	0	17598	0	0	23046	0	0	18930
0.5	32	3	19034	0	0	16920	0	0	16561	13	1	18736	0	0	16218
0.6	32	3	18553	0	0	16907	0	0	14564	0	0	17112	0	0	16443
0.7	11	1	20615	0	0	15160	0	0	13298	0	0	20438	0	0	16866
0.8	0	0	19722	0	0	15287	0	0	14445	0	0	17806	0	0	14925
0.9	0	0	18998	0	0	12965	0	0	12369	0	0	18245	0	0	13479
1.0	0	0	19097	0	0	12128	0	0	12491	0	0	18843	0	0	12046

Table 14: Results for **many relationships** test instances

Table 14 to Table 16 show the results achieved for the generated problem instances using the heuristics. The relationship cost incurred by the various heuristics is shown graphically in Figure 14 to Figure 16.

First we observe how the difference in admissibility density of knapsacks affects the performance of the various heuristics. We see that when the admissibility density of knapsacks is low, the number of items not assigned is high. This is reasonable since each item is only allowed to go into few choices of knapsacks, and thus there will be a higher number of unassigned items. Generally, as the admissibility density increases,

Density	Greedy			Tabu Search			SA			SWO			SWO + Tabu Search		
	Size not assign	#not assign	Relation cost	Size not assign	#not assign	Relation cost	Size not assign	#not assign	Relation cost	Size not assign	#not assign	Relation cost	Size not assign	#not assign	Relation cost
0.1	1679	176	58723	157	33	63045	0	0	64671	1043	106	56074	0	0	61989
0.2	1142	169	61157	0	0	52913	0	0	52347	388	37	56577	0	0	52874
0.3	777	114	61137	0	0	44524	0	0	43880	103	24	53008	9	4	47070
0.4	389	61	63295	0	0	40365	0	0	39033	30	9	50725	0	0	38526
0.5	238	50	65716	0	0	34290	0	0	34606	0	0	45610	0	0	36878
0.6	159	31	66366	0	0	30795	0	0	30650	0	0	41086	0	0	33249
0.7	51	9	64731	0	0	29730	0	0	28521	0	0	38608	0	0	32409
0.8	40	7	62563	0	0	26438	0	0	26677	0	0	34750	0	0	28642
0.9	0	0	60705	0	0	25488	0	0	25862	0	0	32773	0	0	27304
1.0	0	0	60861	0	0	23513	0	0	22128	0	0	32190	0	0	24741

Table 15: Results for **many items, few knapsacks** test instances

Density	Greedy			Tabu Search			SA			SWO			SWO + Tabu Search		
	Size not assign	#not assign	Relation cost	Size not assign	#not assign	Relation cost	Size not assign	#not assign	Relation cost	Size not assign	#not assign	Relation cost	Size not assign	#not assign	Relation cost
0.1	548	5	30067	494	4	29633	494	4	29088	470	2	27065	293	1	27566
0.2	347	2	33927	293	1	29362	293	1	25754	293	1	22910	293	1	22608
0.3	293	1	38778	293	1	27552	293	1	22068	293	1	21066	293	1	20744
0.4	293	1	37111	293	1	24277	293	1	22207	293	1	19986	293	1	19495
0.5	0	0	37130	0	0	24451	0	0	20468	0	0	20330	0	0	19520
0.6	0	0	36690	0	0	24701	0	0	18900	0	0	19827	0	0	19136
0.7	0	0	39192	0	0	21211	0	0	18983	0	0	19068	0	0	18798
0.8	0	0	37714	0	0	21106	0	0	18342	0	0	17357	0	0	17204
0.9	0	0	35953	0	0	20674	0	0	16647	0	0	18945	0	0	18468
1.0	0	0	36109	0	0	19589	0	0	17096	0	0	17824	0	0	16370

Table 16: Results for **few items, many knapsacks** test instances

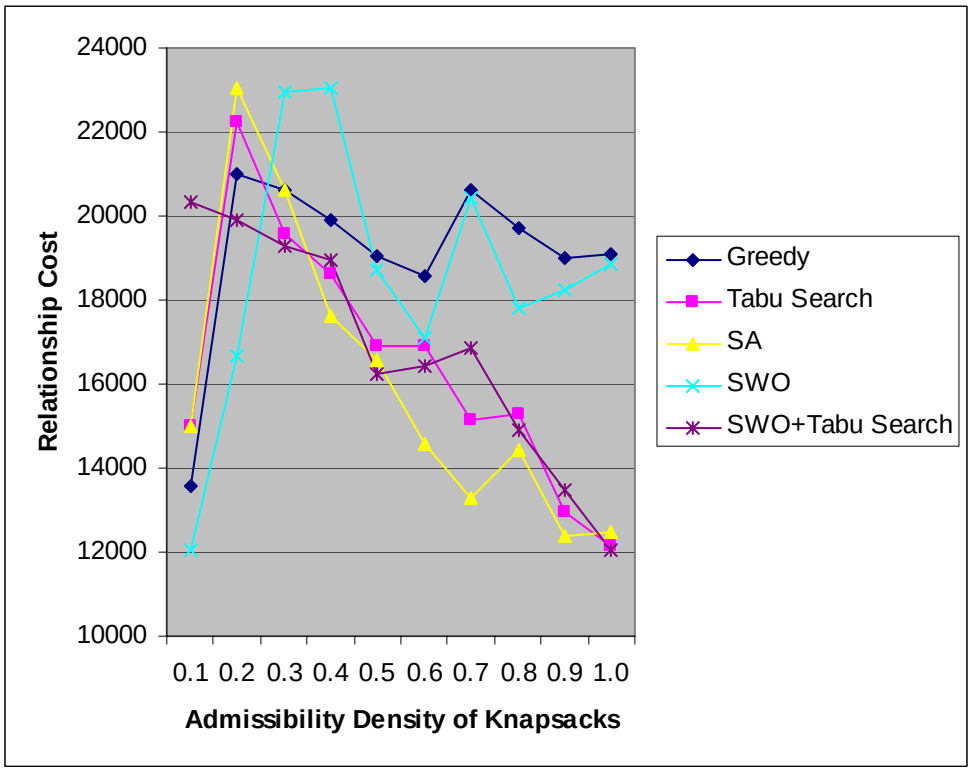

Figure 14: Relationship Cost for **many relationships** test instances

most, if not all, items are able to pack into the available knapsacks. Hence, the heuristics mainly focus on reducing the relationship cost as the admissibility density gets higher. We also see the general trend that as the admissibility density figure gets higher; the relationship cost tends to be lower. Though in some cases, the relationship cost do appear to be quite low when the admissibility density of knapsacks is very low. That's because the restrictive choice of knapsacks that an item may be assigned to has caused the number of items that could not be assigned a knapsack to be high. These items that are not assigned a knapsack do not incur relationship cost. As the choices of knapsacks that are made available to the items increases, the chance of being assigned to a knapsack increases. These items when assigned to a knapsack may result in a

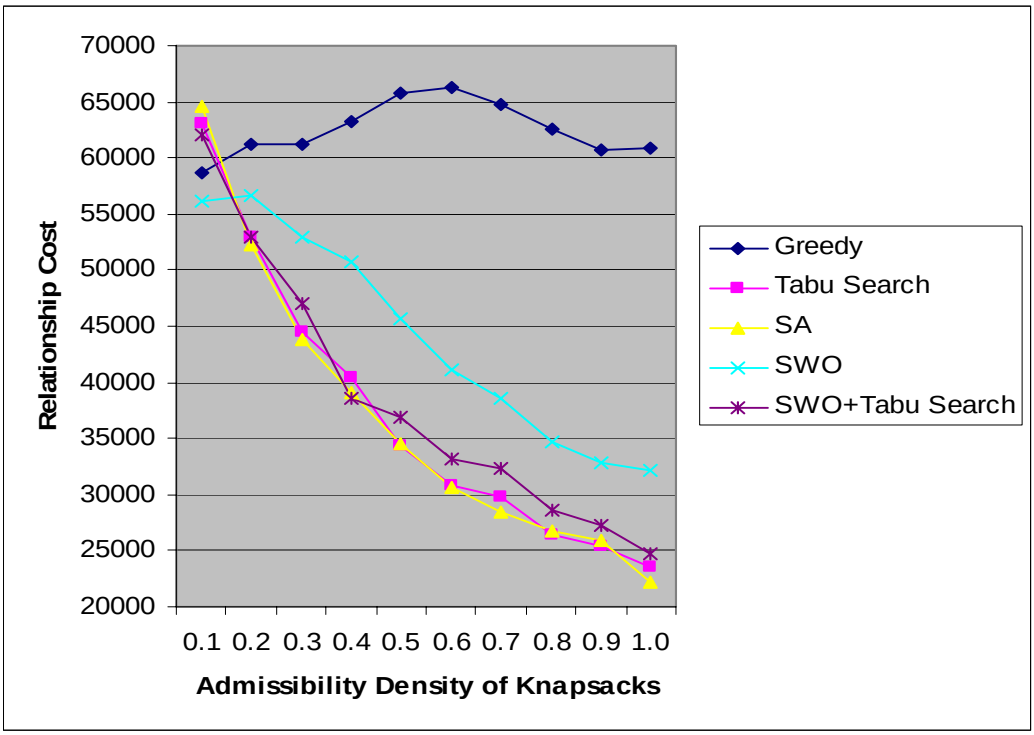

Figure 15 : Relationship Cost for **many items, few knapsacks** test instances

relationship cost if it is assigned to a different knapsack from those items it has a relationship with. This leads to an increase in the overall relationship cost. When there are more choices for the items to be assigned to, each item can basically be assigned to any knapsack as long as it is within the capacity limit. As a result, the chance of matching an item to a less costly knapsack gets higher, causing the overall relationship cost to be lowered. However because of this added flexibility, the solution search space of the problem increases. This in turn increases the computational time required by the search heuristics during their search process as can be seen in Table 17 to Table 19.

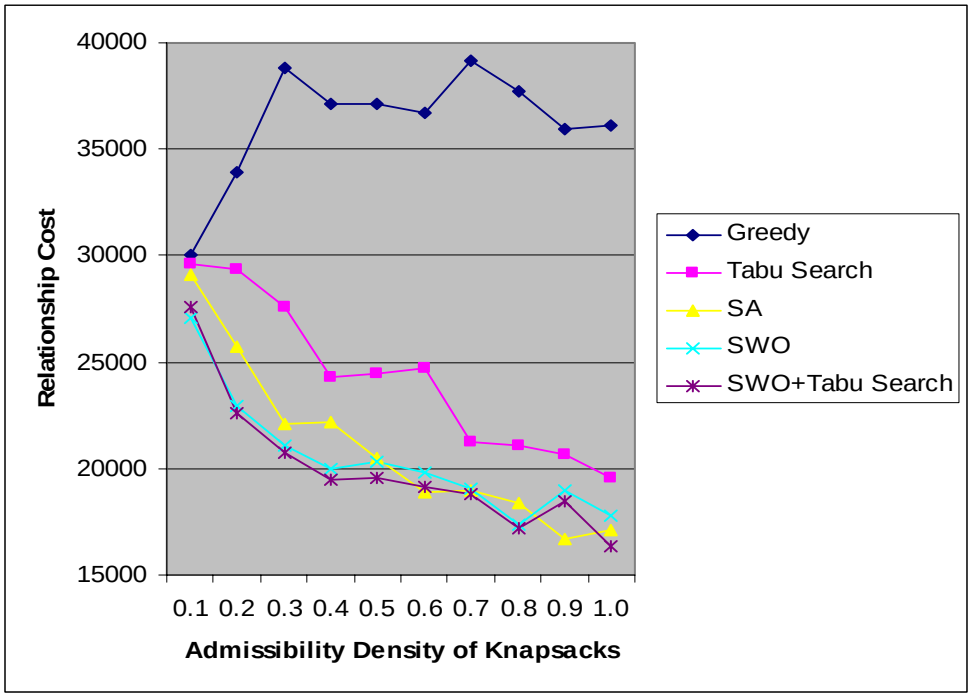

Figure 16: Relationship Cost for **few items, many knapsacks** test instances

Admissibility Density	Greedy	Tabu Search	SA	SWO	SWO + Tabu Search
0.1	1	14	45	23	28
0.2	1	17	22	25	31
0.3	1	19	14	28	35
0.4	1	20	76	31	38
0.5	1	24	64	35	43
0.6	1	21	146	42	49
0.7	1	21	85	43	52
0.8	1	23	74	47	54
0.9	1	25	73	51	60
1.0	1	25	60	50	58

Table 17: Computational Time for **many relationships** test instances

Admissibility Density	Greedy	Tabu Search	SA	SWO	SWO + Tabu Search
0.1	1	1091	599	145	694
0.2	1	1232	617	173	749
0.3	1	1358	618	201	737
0.4	1	1368	588	225	1009
0.5	1	1353	1086	245	990
0.6	1	1354	881	250	943
0.7	1	1564	680	266	1091
0.8	1	1554	1096	286	1217
0.9	1	1423	1370	302	1066
1.0	1	1577	915	311	1276

Table 18: Computational Time for **many items, few knapsacks** test instances

Admissibility Density	Greedy	Tabu Search	SA	SWO	SWO + Tabu Search
0.1	1	47	76	8	15
0.2	1	56	68	11	18
0.3	1	52	63	15	22
0.4	1	48	64	17	24
0.5	1	58	72	21	28
0.6	1	56	79	24	31
0.7	1	53	94	29	36
0.8	1	56	71	32	38
0.9	1	51	84	34	42
1.0	1	68	57	38	45

Table 19: Computational Time for **few items, many knapsacks** test instances

For the **many items, few knapsacks** test instance, we noticed that Tabu Search actually requires substantially more computational time than the rest of the heuristics. This is mainly due to our implementation of Tabu Search in which the tabu operators will find all possible moves before selecting the best move to be applied in every iteration. Hence, in this test instance where there are many items, the large number of possible moves that need to be computed slows down the search process of Tabu Search considerably. Similarly, the computational time required by the hybrid approach also increases for this test instance since Tabu Search is being used as a post-optimization process.

Taking all the results into account, it appears that the hybrid approach performs relatively well in the sense that it is able to efficiently reduce the number and size of items not assigned when the admissibility density of knapsack is low. However as the admissibility density increase, SA appears to perform better although Tabu Search and the hybrid approach appear not to perform too badly either. This may suggest that SA may be good in reducing the relationship cost. One problem with Tabu Search when applied on MKPIR is that it may take a long time when the number of items is huge.

Chapter 6

Conclusions

6.1 Project Summary

In this thesis, we have introduced the multiple knapsack problem with inter-related items, a variant of the multiple knapsack problem that allows assignment restriction and where relationship which exists between items affects the profit of an item in a knapsack. We have also formulated the Venue Assignment Problem, a sub-problem of the examination timetabling problem in NUS as an instance of MKPIR. We have performed a few heuristics on the MKPIR using both actual data from the National University of Singapore and generated test instances and presented our results. Our experimentation shows that the combination of SWO with TS produces promising results in solving MKPIR. To the best of our knowledge, no similar experiment has been conducted on MKPIR or VAP. As can be seen from the example of the venue assignment problem, MKPIR models allocation and packing problems with inter-related items, which is highly relevant in real world applications. We believe that an algorithm that provides an efficient assignment to VAP and the MKPIR will bring great benefits to many other real world problems with similar characteristics.

6.2 Directions for Future Work

In MKP, often, even if the total available size of the knapsacks is greater than the total size of items, it may not be possible to pack all items into the knapsacks. Due

to the nature of the venue assignment problem, when an examination could not be wholly assigned to a venue, it should be allowed to be assigned to multiple venues. This makes the problem to be similar to fractional multiple knapsack problem. However in the VAP, practically the number of examinations that need to be held in multiple venues should be minimal. In addition, the number of venues in which an examination is being split to should be kept to the minimum. Besides causing an increase in manpower, an examination being conducted in multiple venues often leads to confusion for candidates. Candidates may turn up at the wrong exam venue. This differentiates our problem from the classical fractional knapsack problem where the number of splits is not a concern.

For all of our heuristics, we however do not attempt to minimize the number of items that need to be held in multiple knapsacks nor the number of knapsack that the item is being split to within the algorithm. Instead, these items are split to knapsacks that are nearby and that have available capacity after the algorithm terminates. This however does not guarantee that splits are minimized. Allowing an item to be assigned to multiple knapsacks together with the mentioned considerations would add another dimension of difficulty to MKPIR. Which items should be split? How many parts should the item be split into? How big should each part be?

Future efforts could possibly take into consideration the minimizing of splitting an item to multiple knapsacks within the heuristics. One possible approach is dynamic programming.

Our implementation of SWO uses a relatively simple blaming system based on the objectives we seek. When the results of SWO is post-optimized using Tabu Search,

the final results obtained is encouraging. In future works, a more complex blaming system could possibly be employed. Combining the action of SWO with various heuristics that complements the pitfall of each other would be interesting.

Bibliography

Dawande M., P. Keskinocak, R. Ravi and F.S. Salman, *Approximation Algorithms for the Multiple Knapsack Problem with Assignment Restrictions*, Journal of Combinatorial Optimization, Vol. 4, 2000, pg. 171-186

Dawande M. and J. Kalagnanam, *The Multiple Knapsack Problem with Color Constraints*, IBM Research Report, 1998

Garey M. R. and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979

Glover F., *Future Paths for Integer Programming and Links to Artificial Intelligence*, Computer & Operational Research, Vol. 5, 1986, pg. 533-549

Glover F. and M. Laguna, *Tabu Search*, In Reeves (Reeves 1993), chapter 3, 1993

Hung M. S. and J. C. Fisk, *An Algorithm for 0-1 Multiple Knapsack Problems*, Naval Research Logistics Quarterly, Vol. 24, pg. 571-579, 1978

Joslin D. and D. Clement, *"Squeaky Wheel" Optimization*, In Proceedings of AAAI 1998, pg 340-346

Joslin D. and D. Clement, *"Squeaky Wheel" Optimization*, Journal of Artificial Intelligence Research, Vol. 10, 1999, pg. 353-373

Kirkpatrick S., C.D. Gelatt and M.P. Vechhi, *Optimization by Simulated Annealing*, Science, Vol 220 1983, pg 671-680

Lim A., J.C. Ang, W.K. Ho and W.C. Oon, *A Campus-Wide University Examination Timetabling Application*, Innovative Applications in Artificial Intelligence (AAAI/IAAI) 2000, pg. 1020-1025

Lim A., J.C. Ang, W.K. Ho and W.C. Oon, *UTTSExam: A Campus-Wide University Examination Timetabling System*, Innovative Applications in Artificial Intelligence (AAAI/IAAI) 2002, pg. 838-844

Martello S. and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*, 1990, Wiley, Chichester, England

Pisinger D. and P. Toth, *Handbook of Combinatorial Optimization – Knapsack Problems*, Vol. 1, pg 299 – 428, 1998, Kluwer Academic Publishers

Rayward-Smith V. J., I.H. Osman, C.R. Reeves and G.G. Smith, *Modern Heuristic Search Methods*, Wiley, New York, 1996

Reeves C.R., *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell Scientific Publications, Oxford, 1993

Shachnai H. and T. Tamir, *On Two Class-Constrained Versions of the Multiple Knapsack Problem*, Algorithmica, Vol 29, 2001, pg. 442-467

Yang M., *An Efficient Algorithm to Allocate Shelf Space*, European Journal of Operational Research, Vol. 131, 2001, pg. 107-118