AN INVESTIGATION OF MONTE CARLO TREE SEARCH AND LOCAL SEARCH FOR COURSE TIMETABLING PROBLEMS

SAY LENG GOH, MSc.

Thesis submitted to the University of Nottingham
for the degree of Doctor of Philosophy

JULY 2017

Abstract

The work presented in this thesis focuses on solving course timetabling problems, a variant of education timetabling. Automated timetabling is a popular topic among researchers and practitioners because manual timetable construction is impractical, if not impossible, as it is known to be NP-hard.

A two-stage approach is investigated. The first stage involves finding feasible solutions. Monte Carlo Tree Search (MCTS) is utilized in this stage. As far as we are aware, it is used for the first time in addressing the timetabling problem. It is a relatively new search method and has achieved breakthrough in the domain of games particularly Go. Several enhancements are attempted on MCTS such as heuristic based simulations and pruning. We also compare the effectiveness of MCTS with Graph Coloring Heuristic (GCH) and Tabu Search (TS) based methods. Initial findings show that a TS based method is more promising, so we focus on improving TS. We propose an algorithm called Tabu Search with Sampling and Perturbation (TSSP). Among the enhancements that we introduced are event sampling, a novel cost function and perturbation. Furthermore, we hybridize TSSP with Iterated Local Search (ILS).

The second stage focuses on improving the quality of feasible solutions. We propose a variant of Simulated Annealing called Simulated Annealing with Reheating (SAR). SAR has three features: a novel neighborhood examination scheme, a new way of estimating local optima and a reheating scheme. The rigorous setting of initial and end temperature in conventional SA is bypassed in SAR. Precisely, reheating and cooling were applied at the right time and level, thus saving time allowing the search to be performed efficiently. One drawback of SAR is having to preset the composition of neighborhood structures for the datasets. We present an enhanced variant of the SAR algorithm called

Simulated Annealing with Improved Reheating and Learning (SAIRL). We propose a reinforcement learning based method to obtain a suitable neighborhood structure composition for the search to operate effectively. We also propose to incorporate the average cost changes into the reheated temperature function. SAIRL eliminates the need for tuning parameters in conventional SA as well as neighborhood structures composition in SAR.

Experiments were tested on four publicly available datasets namely Socha, International Timetabling Competition 2002 (ITC02), International Timetabling Competition 2007 (ITC07) and Hard. Our results are better or competitive when compared with other state of the art methods where new best results are obtained for many instances.

Acknowledgement

Thanks God for giving me the opportunity to start and finish this challenging journey.

First of all, I would like to thank my main supervisor, Professor Graham Kendall and co-supervisor, Dr. Nasser R. Sabar for their continuous guidance, support and patience. This research project would not have been successful without their assistance.

I would also like to express my sincere gratitude to my external thesis examiner, Professor Salwani Abdullah and internal thesis examiner, Dr. Tomas Maul.

Many thanks to Marcus Gung (Post-graduate Administrator) for his continuous help and advice from the day of registration until the day of thesis submission.

Last but not least, my greatest appreciation to my parents, wife, children and sisters for their understanding, encouragement and moral support throughout the study.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

Combinatorial Optimization Problems (COP) involve finding the values for a set of variables from the discrete search space which maximizes or minimizes an objective function. Examples of these type of problems include vehicle routing problems [90], traveling salesman problems [98], bin packing problems [125], minimal spanning tree problems [91] and timetabling problems [43].

Generally, timetabling is the placement of resources in time. The aim is to maximize profit (which may not be only monetary) where resources are placed in such a way to optimize utilization (thus decreasing operating costs) and satisfy stakeholders' requirements.

In fact, there are many types of timetabling problems, including educational timetabling [155, 104, 11], sports timetabling [105], transportation timetabling [42, 43, 74, 101], nurse rostering [31] etc. We focus on the methods for solving the post enrollment course timetabling problem, a variant of education timetabling. In this context, the timetable is a placement of courses to a finite number of time slots and rooms, satisfying a set of requirements.

In a university setting, modular course structures allow students to choose their preferred courses every semester. Hence, the universal minimum requirement is to ensure that they can attend their lectures without clashing with

other lectures taking place at the same time. Essentially, two events attended by a student cannot be assigned to the same time slot. This scenario is identical to a graph coloring problem where two nodes connected by an edge cannot be assigned the same colour. de Werra shows that timetabling is NP-hard by the reduction of timetabling to graph coloring (known to be NP-hard) [61, 60]. Timetabling construction is also shown to be NP-hard in several other ways [75, 54].

Course timetable construction is indeed difficult as additional requirements further complicate the process. The other common requirements may include the following;

- the room allocated for the course must have enough seating capacity to cater for the number of students enrolled.
- the room allocated for the course must fulfill the features required.
- only one course should be assigned to one room at any time.
- the assignment of courses has to fulfill the precedence requirement e.g. a course should be scheduled to occur before and after certain courses.
- courses can only be assigned to some predefined time.

With these requirements, constructing course timetables manually for hundreds of courses and hundreds (possibly thousands) of students is complicated and time consuming. Usually, course timetabling is automated [20].

It is also a general belief that NP-hard problems cannot be solved optimally by using exact methods in polynomial time (i.e. it is widely believed P $\neq$ NP). A guide to NP-completeness can be found in [103]. As an alternative, heuristic based approximation methods are often used. These methods may not guarantee optimal solutions, but can produce good solutions in reasonable computational times.

It is important to have a feasible course timetable as it will prevent clashes of courses, where lecturers no longer have to conduct extra classes for students with clashing courses and students will be able to attend all the lectures for all the courses enrolled. The lecture rooms will have enough seats for all the enrolled students in addition to all the teaching equipments (OHP, AV, Internet

connection etc.) required by a specific course. A feasible timetable will allow courses (possibly related) to be conducted in a certain order. For example, some courses may have tutorials and require that tutorials be conducted only after the lectures. In addition, a feasible course timetable may cater to busy lecturers who can only conduct courses at certain times. Furthermore, a high quality timetable will prevent students from attending more than two lectures consecutively as students may have limited concentration [167] thus affecting learning effectiveness. An efficient timetable will not require students attending only one lecture on a day, saving commuting costs (time and monetary). Students also prefer not to have lectures in the last slot of the day so that they may have more time for extra curricular activities.

1.2 Aims and Objectives

Monte Carlo Tree Search (MCTS) is a relatively new search method. In recent years, it has become the focus of Artificial Inteligence (AI) thanks to its success and impact in the domain of games, particularly Go. Programs based on MCTS are now competitive with the best human players which was not possible before [115, 73, 58]. While MCTS methods are popular in games decision making, it is rarely used for combinatorial optimization problems. Among the limited number of applications are job shop scheduling by Runarsson et al. [154], one player puzzle by Schadd et al. [158], reentrant scheduling problem by Matsumoto et al. [126] and production management problems by Chaslot et al. [49]. As far as we are aware, MCTS has never been applied to course timetabling problems. The success of MCTS in the games domain motivated us to find out how MCTS would fare in course timetabling problems which are dominated by local search methods, with regard to a solution methodology. In addition, Go and timetabling are similar to some extent in terms of moves. In Go, stones are strategically placed into cells to maximize the number of stones (black or white) on the board, whereas in timetabling, events are strategically placed into cells to maximize the number of assigned events. Another consideration is based on the success of Ant Colony Optimization (ACO) approaches on timetabling. As MCTS and ACO are related in the sense that both are reward based methodologies (reinforcement learning), therefore we believe MCTS will be feasible for timetabling problems as well.

Timetabling algorithms are categorized as one-stage, multi-stage and multi-stage with relaxation. We concentrate on a multi-stage approach where early stages focus on finding feasible solutions while later stages focus on improving the solution quality (soft constraint violations) once a feasible solution is found. We aim to develop an algorithm that is not only applicable to various course timetabling problems with reduced tuning requirement of meta-heuristics but is also competitive or better than the bespoke algorithms found in the scientific literature.

Our objectives are;

- to apply MCTS for the first time in finding feasible solutions for the course timetabling problem and propose enhancements where appropriate.
- to compare the effectiveness of MCTS with classical graph coloring heuristics approach (which is widely used for timetabling problems), as well as the state of the art Tabu Search in finding feasible solutions.
- to identify the weaknesses of the best algorithm in finding feasible solutions and propose enhancements to further improve it.
- to discover and study the state of the art methods applied in reducing the soft constraint violations for course timetabling problems. The objective is to identify drawbacks and propose enhancements.

1.3 Implementation and Computational Experimentation

All algorithms presented in this thesis are coded in the Java Language, using JCreator as an IDE. The experiments are performed on windows server 2008, Intel Xeon (3.1 GHz) with 4Gb RAM machines. The computation time limit allowed by running the benchmark program[1] is T=190 seconds for each single run. Each run will stop when the runtime equivalent to T is reached unless specified otherwise. A total of n=31 runs were executed for each instance so that we could carry out a statistical analysis.

[1] http://www.idsia.ch/Files/ttcomp2002/ Last accessed: June 13, 2017.

1.4 Scientific Publications

The research reported in this thesis has produced the following publications.

- Say Leng Goh, Graham Kendall, Nasser R. Sabar. Improved local search approaches to solve post enrolment course timetabling problem. European Journal of Operational Research, 2017.
- Goh SL, Kendall G, Sabar NR. [date unknown]. A Hybrid Local Search Approach for Hard Course Timetabling Problem. Information Systems and Operational Research (INFOR). Under review.
- Goh SL, Kendall G, Sabar NR. [date unknown]. Solving Post Enrolment Course Timetabling (PE-CTT) Problem using Simulated Annealing with Improved Reheating and Learning (SAIRL). Journal of Operational Research Society (JORS). Under review.

1.5 Thesis Structure

We have presented the background, motivation, aims, objectives, computational experimentation and scientific publications. The rest of the thesis is organized as follows. Chapter 2 provides a review on the educational timetabling (variants and solution approaches) as well as the utilization of MCTS for combinatorial optimization problems. Chapter 3 gives a review on course timetabling such as problem description, formal representation and specific approaches utilized on the datasets considered in this thesis. In chapter 4, we compare MCTS with several other approaches in finding feasible solutions. Chapter 5 outlines several enhancements proposed on TS (the method with the best potential in finding feasible solutions). Chapter 6 is dedicated to improve the quality of feasible solutions in terms of soft constraint violations. The SAR algorithm is proposed. In Chapter 7, we further enhance the SAR algorithm in terms of ease of use and performance. The SAIRL algorithm is proposed. Chapter 8 wraps up the thesis where thesis summary and future work are presented.

Chapter 2

Literature Review: Educational Timetabling

2.1 Introduction

As manual timetable construction is a daunting task and not practical due to its complexity, automated timetabling has attracted the focus of researchers and practitioners alike. This chapter presents the related work in automated timetabling. The key introduction and surveys on educational timetabling can be found in [61, 160, 38, 106, 29, 141].

2.2 Variants of Educational Timetabling

Education timetabling is a variant of timetabling and it can be further classified into three main classes [160]. They are similar in some ways yet different mainly in terms of stakeholders and constraints involved. The classes are:

2.2.1 School Timetabling

School timetabling is the allocation of class, teacher and room of a school on weekly basis. Students are usually grouped into classes before timetable construction. The school timetabling problem differs in terms of requirements of the class, teacher and room. The universal aim is to prevent teachers from attending two or more classes simultaneously. An overview of school timetabling

can be found in [142]. The third timetabling competition (ITC11) focuses on the school timetabling problem where 35 of the instances were taken from schools in 10 countries [143].

2.2.2 Course Timetabling

This involves the scheduling for all lectures of a university on a weekly basis aiming to prevent students from attending two or more lectures at the same time. Only one lecture can be conducted per room at the same time. Usually time slots are fixed for the week. Surveys on university timetabling can be found in [47, 127].

2.2.3 Examination Timetabling

Examination timetabling is the scheduling for all exams of a university aiming to prevent students from attending two or more exams at the same time. Usually, multiple exams can be scheduled in a room at the same time provided the seating capacity is not exceeded. Time slots are not fixed as in course timetabling. Students prefer to spread the exams so that they may have enough time for revision and rest. Another common requirement is for the large exams to be placed early to allow more time for marking. Surveys can be found in [44, 46, 28, 145].

2.3 Graph Colouring Model of Timetabling

Timetabling problems are closely related to graph colouring problems. Given a simple and undirected graph, G=(V, E), where V is a set of vertices and E is a set of edges joining the vertices, the graph colouring problem involves assigning minimal colours (chromatic numbers) to the vertices such that vertices joined by common edges are assigned different colours. Finding the chromatic number is NP-hard. An analogy can be drawn between graph colouring and timetabling where events correspond to vertices, clashes between events are represented by the edges, while time slots correspond to colours. In timetabling, we try to assign clashing events to different time slots. Despite having similarities, timetabling problems are more complex with additional constraints and a mere reduction to graph colouring problem is insufficient. However, heuristics extracted from graph colouring serve as a basis for researchers in solving timetabling problems.

The heuristics used to order the vertices for colouring a graph are applied to timetabling to order the events for slot assignment. Among the common heuristics used are;

- Largest Degree (LD): events are ordered based on the number of clashes with other events. Events are clashing when they are attended by the same student. Usually, events with a higher clash count are harder to schedule and therefore should be considered first.
- Saturation Degree (SD) [23]: the next event to be scheduled is the one with the lowest remaining time slots. It is calculated dynamically one event after another. Ties are broken on largest degree.
- Largest Weighted Degree (LWD): events are ordered based on largest degree weighted by student enrolment.
- Largest Enrolment (LE): events are ordered by the number of students registered for the events. Usually, events with more students tend to have more clashes with other events, thus harder to schedule and therefore should be considered first.
- Colour Degree (CD): events are ordered by the number clashes with those already scheduled. It is calculated dynamically after each event is scheduled.
- Random Ordering(RO): events are ordered randomly.

Sabar et al. [156] utilized graph colouring heuristics together with a constructive hyper-heuristic for examination timetabling problems. The most difficult exam with the minimum difficulty index (calculated by using hierarchical hybridizations of four graph colouring heuristics namely LD, SD, CD and LE) was assigned to time slots selected probabilistically by roulette wheel selection, where time slots with smaller penalties were more likely to be selected.

Another important concept derived from graph colouring which could be useful is the identification of cliques. A clique is a collection of vertices that are mutually adjacent. Vertices in V are adjacent if they are end vertices of some edge in E. All vertices in a clique must be assigned different colours. Therefore, if the maximum clique size of a graph colouring problem instance is known to be x, then x is minimum colors needed to colour the graph. However, finding

the maximum clique size is NP-hard. Similarly, all events in a clique have to be assigned to different time slots and if the maximum clique size can be determined, it will give us a hint on the minimum number of time slots needed to assign all the events.

2.4 Timetabling Approaches

Many approaches have been applied to timetabling, including graph colouring based heuristics [26], constraint programming techniques [177] and integer programming [59]. Lately, meta-heuristics approaches have been popular among researchers due to its success. A survey by Lewis on meta-heuristics and timetabling shows that algorithms can generally be classified into three categories namely, one-stage, multi-stage and multi-stage with relaxation [119].

2.4.1 One-Stage

Algorithms of this category attempt to find solutions satisfying both hard and soft constraints simultaneously. The approach is achieved through a weighted sum function of constraints. Having a single objective function is beneficial as it is applicable to any optimization technique. The drawback is, it is difficult to set the right weight for each constraint. In static weight setting, hard constraints are usually given higher penalty than soft constraints which may limit the search space navigation. Alternatively, dynamic weight setting is used where appropriate adjustments are made at runtime. Schaerf employed adaptive relaxation in solving high school timetabling problems [159]. The weights of the hard constraints were increased when all previous k moves resulted in infeasible solutions and decreased when all k moves produced feasible solutions. The weights were left unchanged when both feasible and infeasible solutions existed in the k moves. When respecting hard constraints is of primary concern, the search will be less effective as it is distracted by the soft constraints component. Furthermore, feasibility is not guaranteed at the end of the search. This approach is suitable for problems where achieving feasibility is easy and tolerable.

2.4.2 Multi-Stage

In the initial stage, only hard constraints are considered. Once feasibility is achieved, the focus is shifted towards optimizing soft constraints without compromising feasibility. Thomson and Dowsland applied this approach in addressing examination timetabling problems [171]. Feasibility was obtained using graph colouring heuristics before applying Simulated Annealing to minimize the soft constraint violations. One obvious advantage is the need for weights balancing among the hard and soft constraints is eliminated. This approach is suitable for problems where achieving feasibility is compulsory for the solution to be usable as the search for feasibility is focused. Gogos et al. utilized a multi-stage algorithmic process (resembling GRASP) for the examination timetabling problem [87]. There were six stages in their approach namely preprocessing, feasibility, Hill Climbing, Simulated Annealing, Integer Programming (IP) sub-problem and Shaking. Shaking was invoked only when the IP sub-problem stage failed to improve the current solution. Their method was ranked second among competitors in ITC07 (examination timetable track).

2.4.3 Multi-Stage Relaxation

In this approach, some aspects of the problem such as feasibility are relaxed when optimizing certain hard and soft constraints. Relaxation allows better connectivity of the search space, flexibility and prevents obstacles to achieve good quality solutions. This algorithmic approach comprises multiple stages. Some constraints are relaxed while satisfying other constraints initially where relaxation is then removed later. For example, unassigned events are left aside while attempting to satisfy soft constraints. Attempts are made to insert the unassigned events later. Some researchers introduced extra time slots to satisfy hard and soft constraints and reduced the additional slots later in the processing. Cambazard et al. applied the colouring relaxation strategy on course timetabling [41]. There were four stages in their implementation. First, courses were assigned to time-slots ignoring room allocation. Second, the soft constraint violations of the solution were improved. Third, the room relaxation was fixed. Forth, the soft constraint violations were again improved.

2.5 Meta-heuristics

Heuristic is defined by Reeves [147] as a

> *Method which seeks good solutions at a reasonable computation cost without guarantee on optimality and feasibility.*

As heuristics are usually problem dependent, a general guide for the heuristics is needed to cater for different problems. The term meta-heuristic first used by Glover and defined by him as [33]

> *The master strategy that guides and modify heuristics to produce solutions beyond those normally generated in a quest for local optimality..*

Osman and Kelly [66] define meta-heuristic as

> *An iterative process guiding heurictics to explore and exploit the search space to find near optimal solutions.*

Another definition is provided by Metaheuristics Network (MN), a european research project as

> *General algorithmic framework applicable to different optimization problems with relatively few modifications.*

There are five main metaheuristics paradigms mentioned by MN namely; Evolutionary Algorithm (EA), Ant Colony Optimization (ACO), Tabu Search (TS), Simulated Annealing (SA) and Iterated Local Search (ILS). Rossi-Doria et al. compared the performance of different metaheuristics on the university course timetabling problem [153].

2.5.1 Local Search Based

The solution space is explored by iteratively deciding to replace the current solution with its neighbour with the aim of finding the best solution.

2.5.1.1 Steepest Descent (SD)

Steepest Descent (SD) or best improvement local search is shown in Algorithm 1 (adapted from [33]). All the neighbors of the current solution are explored

completely. The search is stopped when the candidate solution is worse than the current.

Algorithm 1

```
1: procedure STEEPEST DESCENT
2:     current ← initial solution
3:     repeat
4:         candidate ← arg min_{x∈N(current)} f(x)
5:         if f(candidate) < f(current) then
6:             current ← candidate
7:         end if
8:     until f(candidate) >= f(current)
9: end procedure
```

2.5.1.2 First Descent (FD)

As SD is time consuming by having to explore all the neighbors completely, First Descent (FD) serves as an alternative. FD is presented in Algorithm 2 (adapted from [33]). The first candidate neighbour better than the current solution will replace the current solution. The search stops when none of the neighbors of current is better than the current solution.

Algorithm 2

```
1: procedure FIRST DESCENT
2:     current ← initial solution
3:     i ← 1
4:     repeat
5:         candidate ← x_i ∈ N(current), i = 1, ..., i_max
6:         if f(candidate) < f(current) then
7:             current ← candidate
8:             i ← 1
9:         else
10:            i ← i + 1
11:        end if
12:    until i = i_max
13: end procedure
```

2.5.1.3 Tabu Search (TS)

TS was introduced by Glover [85] as an extension to hill climbing to overcome local optima. TS utilizes a memory to avoid reversal of recent moves. TS is implemented by maintaining a list of forbidden moves. After each move, the re-

versal move is added to the end of list while the oldest entry is deleted. The tabu tenure, indicated by the length of the list, determines the number of iterations for which the reversal moves remain forbidden. Consequently, some attractive moves may be prohibited even if they do not incur cycling. An aspiration criteria alleviates the problem by conditionally allowing a tabu move even if it is tabu. The simplest case is allowing a tabu move if the resulting solution is better than the best solution seen so far. Best improvement is the most common variant of TS which selects the best move (non-tabu or allowed by aspiration) regardless of whether the move is improving or not [33]. This allows the search to explore new areas and escape from local optima. A basic TS algorithm is shown in Algorithm 3 (adapted from [84]). As evaluating every neighbour of the current solution incurs a computational overhead, only a random sample of the neighborhood is considered in probabilistic TS. Glover and Laguna [86] mentioned two important strategies to make the search more effective, namely intensification and diversification. Intensification involves thoroughly exploring the promising search space by restarting the search from the currently best known solution and freezing the attractive components. Intensification can also be applied by increasing sampling size or changing the neighbourhood structure [33]. Diversification involves examining the rarely or unvisited regions of search space.

Algorithm 3

```
1: procedure Tabu Search
2:     Tabu list T
3:     current ← initial solution
4:     best ← current
5:     repeat
6:         current ← arg min_{x∈N(current)} f(x), x is non-tabu
7:         if current < best then
8:             best ← current
9:         end if
10:        record the recent move in T (delete the oldest entry if necessary)
11:    until stop condition
12: end procedure
```

TABUCOL is one the best implementations of TS for the graph colouring problem [97]. It is based on improper k-colouring approach where all vertices were coloured but may contain clashes. At each iteration, a set of neighbors were generated by randomly choosing a vertex (with colour i) among the clash-

ing ones and randomly assigning a colour j where $i \neq j$. The best neighbour was determined among the neighbors which were non-tabu or allowed by aspiration. The best neighbour was applied and a list was maintained where the original colour was forbidden for the vertex for a certain number of iterations. Variants of TABUCOL are often used as local search procedure in evolutionary hybrid heuristics [65, 80, 78].

Another implementation of TS in graph colouring problems is termed as PARTIALCOL. It uses a partial k-colouring approach where some vertices are coloured and have no clashes. An initial solution was generated using a greedy algorithm. A neighbour solution was obtained by colouring an uncoloured vertex u with a colour. As a result, some vertices might clash with the vertex u and are removed and placed in the uncoloured list. The cost function was defined as the number of vertices in the uncoloured list. The best neighbour was applied and a tabu list was maintained. The author also introduced a new reactive tabu tenure scheme called FOO-scheme. The tenure was increased when the search was trapped (as indicated when the objective function was stagnant or changed little for some time) and decreased slowly along the search to alternate between intensification and diversification.

Schaerf applied TS in high school timetabling problems [159]. An atomic move consisted of swapping lectures of two different periods or transferring a lecture to a different period. A double move comprised of a pair of atomic moves, where the second move repaired the infeasibility created by the first. Randomized Non-Ascendant (RNA) was used to generate an initial good solution for TS. When TS had shown no improvements for a given number of iterations, RNA was run on the best solution to shake up the solution before TS was restarted. The cycle was repeated. The author used the simplest aspiration criteria, where a tabu move was conditionally accepted only if it improved the best solution.

2.5.1.4 Great Deluge (GD)

This algorithm is introduced by Dueck [70]. It accepts all solution with a cost function below or equivalent to a certain level (defined as the *water level*). The level is decreased during the search and bounds the region of the search space. Worse solutions below the boundary are accepted and this allows the search to escape from local optima. Meanwhile, solutions with cost function above the

boundary are always rejected.

Algorithm 4

```
1: procedure Extended Great Deluge
2:     current ← initial solution
3:     level ← f(current)
4:     Δlevel ← (f(current) - expected solution quality) / search time
5:     repeat
6:         candidate ∈ N(current)
7:         if f(candidate) <= f(current) OR f(candidate) <= level then
8:             current ← candidate
9:         end if
10:        level ← level − Δlevel
11:    until stop condition
12: end procedure
```

Burke et al. applied GD in examination timetabling [37]. The water level was decreased by using a decay rate which was computed as the initial solution quality multiplied by a factor (provided by the user) and divided by the number of iterations. The method outperformed Hill Climbing, Simulated Annealing, Tabu Search and graph colouring heuristics.

Burke et al. proposed an extended version of GD on examination and course timetabling problems where candidate solutions, which were better than the current one, were also accepted [27]. The extended GD is shown in Algorithm 4. The method reported superior results compared to other methods.

2.5.1.5 Simulated Annealing (SA)

Metropolis [131] introduced the Metropolis algorithm to simulate the evolution of solid in a heat bath to thermal equilibrium. Kirkpatrick applied the concepts of annealing to optimization problems [107]. SA is presented in Algorithm 5 (adapted from [1]). It accepts all improving moves or those equivalent to the current solution. It also accepts worse moves with probability of:

$$e^{\Delta f/T} \tag{2.1}$$

where Δf is the change in solution quality and T is the temperature. Usually, T is initialized with a sufficiently high value and gradually reduced as the search progresses. In practice, the search ends when the temperature exceeds a prede-

fined end temperature or another stopping condition is met. Generally, larger deteriorations are accepted with lower probability at any temperature. The same deterioration is accepted with higher probability at higher temperatures. Acceptance of deteriorating moves enables the algorithm to escape from local optimum. SA converges asymptotically to a global optimum solution [33]. One drawback of SA is that, it usually requires parameter tuning (initial temperature, end temperature, decay rate and Markov chain length) in order to obtain good quality results.

Algorithm 5

```
1: procedure Simulated Annealing
2:     current ← initial solution
3:     t ← initial temperature
4:     l ← initial length
5:     repeat
6:         for i = 1 to l do
7:             candidate ∈ N(current)
8:             if f(candidate) <= f(current) then
9:                 current ← candidate
10:            else
11:                if exp(f(current) − f(candidate)/t) > random[0,1) then
12:                    current ← candidate
13:                end if
14:            end if
15:        end for
16:        update l
17:        update t
18:    until stop condition
19: end procedure
```

Triki et al. studied the behavior of Simulated Annealing [172]. The classical proposals for SA parameters were discussed. The authors remarked that temperature decrease in optimal annealing (covergence to optimal solution) [93] were too slow and only of theoretical interest. They also demonstrated several classical adaptive cooling schedules proposed in [178, 149, 137] are equivalent and proposed a new adaptive cooling schedule which performed better than the best known schedule for TSP.

Suman and Kumar presented a survey of Simulated Annealing as a tool for single and multi-objective optimization [168]. They reviewed variants of the

SA algorithm and its applications. The annealing schedule, such as initial temperature and the cooling schedule were discussed. Suggestions for performance improvement and future research of SA were also presented.

Battistutta et al. utilized a single stage feature based tuning of SA for examination timetabling [21]. In preliminary tuning, standard SA parameters were tuned using F-Race. In addition, the coefficients of features (eg. students, rooms etc.) were determined, by building a linear regression model in R, and used to predict the ideal value for the hard constraints weight. The authors noted that the method could not always find a feasible solution for the harder instances. Better results were reported compared to the finalist in ITC07. However, they were inferior to the results by Bykov and Petrovic [40].

Thompson and Dowsland applied simulated annealing for the examination timetabling problem [170]. The authors mentioned the difficulty of setting weights in a single phased method. Therefore, a multi-phased method was used where more important objectives were considered in earlier phases, while other objectives were considered in later phases. The optimized objectives in the early phases were considered as binding constraints in later phases. The authors note that their method is not perfect as the solution space may be disconnected. Three ways were introduced to deal with solution space connectivity, but only two were deemed successful, namely; using different starting solutions and changing the neighborhood structure (Kempe chain operator).

Abramson, Amoorthy and Dang compared two cooling schedules (geometric cooling and multiple cooling) and four reheating schemes (geometric reheating, enhanced geometric reheating, non-monotonic cooling and reheating as cost function) on randomly generated school timetabling problems [10]. Cost based reheating was found to be superior in finding global minima and also faster compared to other schemes.

Some other successful implementation of SA were in school timetabling problems [9], course timetabling problems [109, 52, 110, 41, 53, 117, 48, 118] and examination timetabling problems [171].

2.5.1.6 Iterated Local Search (ILS)

ILS is shown in Algorithm 6 ([123]). The current solution is perturbed producing an intermediate solution. The intermediate solution is further processed by a local search function generating a candidate solution. The candidate solution will be set as the current solution if it is accepted by the acceptance criterion. Otherwise, the current solution remains unchanged. The acceptance criterion can be of any type e.g. Hill Climbing, Simulated Annealing, random walk and restart from a new solution after certain number of idle iterations. Lourenco et al. [123] defines a good perturbation as one that transforms a solution into an excellent starting point for a local search. If the perturbation is too strong, ILS behaves like random restart. Meanwhile if perturbation is too weak, the search cannot explore effectively. Ideally, the neighborhood used in the perturbation should be of higher order than the one used in the local search.

Algorithm 6

```
1: procedure ITERATED LOCAL SEARCH
2:     current ← initial solution
3:     repeat
4:         intermediate ← PERTURB(current)
5:         candidate ←LOCALSEARCH(intermediate)
6:         current ←ACCEPTANCECRITERION(current, candidate)
7:     until stop condition
8: end procedure
```

Fonseca et al. [79] hybridized Simulated Annealing with ILS and became the winner of ITC11. The Kingston High School Timetabling Engine (KHE) was used to generate a feasible solution which was further improved by a SA-ILS approach.

2.5.1.7 Variable Neighbourhood Search (VNS)

VNS is a metaheuristic which exploits the idea of neighbourhood change in descent to local minima and being able to escape from it [94]. It is simple and requires few or no parameters. It works based on the following:

- A local minimum for one neighbourhood structure may not be for another.
- A global minimum is a local minimum for all neighbourhood structures
- Local minima are relatively close to one another.

Variable Neighborhood Descent (VND) is an extension to SD. VND is shown in Algorithm 7 (adapted from [33]). Multiple neigborhood structures are considered one by one. If the candidate solution improves the current solution, the first neighborhood structure will be considered in the next iteration. Otherwise the next neighborhood structure will be considered. The iteration stops when all the neighborhood structures are taken into account.

Algorithm 7

```
1: procedure VARIABLE NEIGHBORHOOD DESCENT
2:     neighborhood structures N_k, k = 1, ..., k_max
3:     current ← initial solution
4:     k ← 1
5:     repeat
6:         candidate ← arg min_{x∈N_k(s)} f(x)
7:         if f(candidate) < f(current) then
8:             current ← candidate
9:             k ← 1
10:        else
11:            k ← k + 1
12:        end if
13:    until k = k_max
14: end procedure
```

In Reduced Variable Neighbourhood Search (RVNS) as shown in Algorithm 8 (adapted from [95]), a candidate solution is chosen randomly from a neighborhood structure. If it is better than the current solution, the next candidate solution is generated from the first neighborhood structure. Otherwise, the next neighborhood structure will be considered. After all neighborhood structures have been considered, the search restarts with the first neighborhood structure until the stopping condition is reached.

Algorithm 8

```
1: procedure Reduced Variable Neighbourhood Search
2:     neighborhood structures N_k, k = 1, ..., k_max
3:     current ← initial solution
4:     repeat
5:         k ← 1
6:         repeat
7:             candidate ∈ N_k(current)
8:             if f(candidate) < f(current) then
9:                 current ← candidate
10:                k ← 1
11:            else
12:                k ← k + 1
13:            end if
14:        until k = k_max
15:    until stop condition
16: end procedure
```

Basic Variable Neighbourhood Search (BVNS) is shown in Algorithm 9 (adapted from [95]). It is similar to VNS. Local search is applied on the random candidate solution to obtain the local optimum. When VND is used as the local search, it is called General Variable Neighborhood Search (GVNS).

Algorithm 9

```
1: procedure Basic Variable Neighbourhood Search
2:     neighborhood structures N_k, k = 1, ..., k_max
3:     current ← initial solution
4:     repeat
5:         k ← 1
6:         repeat
7:             candidate ∈ N_k(current)
8:             localOptimum ← localSearch(candidate)
9:             if f(localOptimum) < f(current) then
10:                current ← d
11:                k ← 1
12:            else
13:                k ← k + 1
14:            end if
15:        until k = k_max
16:    until stop condition
17: end procedure
```

Abdullah et al. applied this approach on course timetabling problem in [3, 5]. Di Gaspero and Schaerf employed a multi-neighbourhood local search on course timetabling in [62, 81].

2.5.2 Evolutionary Based

Genetic Algorithm and Ant Colony Optmization are presented here.

2.5.2.1 Genetic Algorithms (GA)

The work of Fraser [13] and Bremermann [24] laid the foundation of genetic algorithms (GA). Seminal work on GAs and its application can be found in [89, 99]. In a GA, many terms are borrowed from its biological inspiration. Chromosomes are solutions, genes are variables and alleles refer to values of those variables. Solutions to search problems are evolved using the following steps:

- Population: candidate solutions of certain population size p are created randomly or heuristically.
- Selection: candidate solutions are evaluated and selected where better solutions are preferred. Among the common selection methods are roulette wheel, ranking and tournament.
- Recombination (crossover): Better solutions (offspring) are created from combination of parts (traits) of parents.
- Mutation: Solution is randomly modified locally. Mutation diversifies the population.
- Replacement: The parental population is replaced by the offspring created using replacement methods such as elitist, steady state etc.

The basic GA is shown in Algorithm 10 (adapted from [157]).

Algorithm 10

```
1: procedure Genetic Algorithm
2:     Population
3:     repeat
4:         Selection
5:         Recombination (crossover)
6:         Mutation
7:         Replacement
8:     until stop condition
9: end procedure
```

Ross et al. succesfully applied GA on course timetabling [150]. Ross et al. investigated examination timetabling using a GA [151]. Abdullah et al. employed memetic algorithm, or simple combination of GA with local search, on course timetabling problem in [4]. Rossi-Doria and Paechter presented a memetic algorithm for university course timetabling [152]. Turabieh and Abdullah incorporated tabu search into a memetic approach in solving course timetabling problems [174]. Silva and Obit tested non-linear great deluge with GA on course timetabling problems [114].

2.5.2.2 Ant Colony Optimization (ACO)

A double bridge experiment inspired Dorigo et al. to apply the first ACO algorithm to the Traveling Salesman Problem (TSP)[64]. In the biological experiment, two branches of different length were connected between the ants and the food source. It is interesting to observe that the ants took the shorter route after a while. The explanation for this was that the ants laid pheromone along the path to guide other ants. The shorter route was preferred as more ants followed it. It had a larger concentration of pheromone as more trips were made by the ants using the shorter route. ACO is presented in Algorithm 11 ([130]). It is an iterative process where ants in each iteration try to find a good solution through a constructive method. The ant will mark the path (construction graph) with pheromone to guide other ants in the following iterations. The pheromone of good solutions are increased. Meanwhile, the pheromone is also evaporated (decreased) over time to prevent the search becoming too restrictive.

Algorithm 11

```
1: procedure Ant Colony Optimization
2:     Initialize pheromone values
3:     repeat
4:         for ant = 1 to m do
5:             construct a solution
6:         end for
7:         for all pheromone values do
8:             decrease the value                          ▷ evaporation
9:         end for
10:        for all pheromone values of good solutions do
11:            increase the value
12:        end for
13:    until stop condition
14: end procedure
```

Application of ACO can be found in graph colouring problems [56, 68]. The approach was also applied in examination scheduling problem [67] and course timetabling problems [164, 71, 102, 135].

2.6 Hyper-heuristics

Hyper-heuristics aim to provide a general, fast, simple and understandable algorithm for a range of problems. They can be thought of as heuristics to choose heuristics. They work on a search space of heuristics instead directly with the solution search space. Hyper-heuristics provide an alternative to meta-heuristics which are often criticized for intensive parameter tuning, resource intensive development, incomprehensible solution arrival and being relatively slow. Interested readers can refer to [30, 32, 138].

Burke et al. proposed a method called Tabu Search Hyper-Heuristic, which was applied to course timetabling [35]. Burke et al. [36] presented a Graph Based Hyper-Heuristic approach for educational timetabling problems. In each iteration, two events were ordered by a list of heuristics and scheduled. If feasible, the list of heuristics was set tabu for 9 iterations, otherwise the list was updated as a failed list. At the end of each iteration, steepest descent was used to improve the solution (exploitation step). The next list was produced by randomly changing two of the heuristics in the previous list. If the new list was not a failed list and non-tabu, the list was used to schedule another two events.

The steps were repeated for a predefined number of iterations.

Bai et al. investigated the effect of memory length towards the performance of Simulated Annealing Hyper-heuristic on course timetabling problems [18]. The algorithm with learning performed better than its counterpart without learning. Short term memory also performed better than a long term memory based algorithm. However, the authors remarked that it was difficult to determine the learning rate that performed best for all problem instances. Static and dynamic (random and incremental) memory length were compared. Results showed that dynamic incremental memory length was not only comparable to the best static learning rate but also did not require parameter tuning as the case for static learning.

Burke et al. investigated Monte Carlo hyper-heuristics for examination timetabling [34]. Several combination of heuristic selection (simple random (SR), greedy (GR), choice function (CF), learning scheme (L)) and Monte Carlo move acceptance (MC, SA, EMCQ) were compared. CF-SA performed the best among the combinations. The authors remarked that L-SA performed poorly due to weakness of rewarding mechanism and evaluation of utility values for heuristic selection.

Soria-Alcaraz et al. utilized an iterated local search based hyper-heuristic approach to address the course timetabling problem [166]. Two types of learning were compared for operator selection, namely static (training and execution) and dynamic (learns on the fly). The dynamic scheme was reported to be better than the static counterpart.

2.7 Monte Carlo Tree Search (MCTS)

Conceptually, MCTS is a tree built incrementally in an asymmetric manner based on samplings of the decision space. The tree is traversed in a best first manner. A decision is made at each state selecting actions based on the values of the nodes, balancing exploitation of good looking moves and exploration of bad looking moves (which may turn out to be good in the long run). Unvisited nodes are added to the tree. A simulation is run towards the end and the outcome is updated on the traversed nodes. The effect is twofold where the tree

traversal guides the simulations and improves the quality of sampling while the selective sampling guides the tree building and traversal.

Each node in the tree represents a state and each directed link represents an action leading to the state. Each node stores information on the average value and visit count. It generally consists of four steps iterated until a stop condition is reached [25, 51]. The four steps are:

- Selection: The tree is traversed from the root until an expandable node is reached (non-terminal and has unvisited children).
- Expansion: Child nodes are added to the tree, according to available actions. One or more nodes can be added depending on memory limitations.
- Simulation: Simulation is run until a terminal state is reached to produce a result.
- Backpropagation: The result is propagated through the traversed nodes where statistics of the nodes are updated.

The general MCTS is shown in Algorithm 12 (adapted from [25]). The sub-procedures are presented in Algorithms 13-18.

Algorithm 12

```
1: procedure General MCTS
2:     Create root node, node_root
3:     while stop condition do
4:         node_last ← treeGrowth(node_root)
5:         reward ← simulation(state(node_last))
6:         backpropagation(node_last, reward)
7:     end while
8:     return bestChild(node_root)
9: end procedure
```

Algorithm 13

```
1: procedure treeGrowth(node)
2:     while node is non-terminal do
3:         if node not fully expanded then
4:             return expansion(node)
5:         else
6:             node ← selection(node)
7:         end if
8:     end while
9:     return node
10: end procedure
```

Algorithm 14

```
1: procedure expansion(node)
2:     create a child node, node_a for a ∈ action(node)
3:     return node_a
4: end procedure
```

Algorithm 15

```
1: procedure selection(node)
2:     return arg max_{i∈children of node} value_i + B sqrt(ln visit_p / visit_i)
3: end procedure
```

Algorithm 16

```
1: procedure simulation(state)
2:     while state is non-terminal do
3:         choose a ∈ action(state)
4:         state ← f(state, a)
5:     end while
6:     return reward for state
7: end procedure
```

Algorithm 17

```
1: procedure BACKPROPAGATION(node, reward)
2:     while node is not null do
3:         update node.visit
4:         update node.value according to reward
5:         node ← parent of node
6:     end while
7: end procedure
```

Algorithm 18

```
1: procedure BESTCHILD(node)
2:     return arg max_{i∈children of node} value_i
3: end procedure
```

MCTS was first coined by Coulom who proposed to combine Monte Carlo evalution with tree search [57]. Simulations were iteratively and randomly run from the root. Nodes close to the root were memorized and updated with values. As the number of simulations increase, the probability of selecting a move at random is changed as moves with higher values were selected more often. In his Crazy Stone's algorithm, which won the 10th Kiseido Go Server (KGS) computer Go tournament, nodes were generally selected based on the probability of being better than the current best. Each move was selected with probability proportional to

$$\exp(-2.4\frac{v_0 - v_i}{\sqrt{2(\sigma_0^2 + \sigma_i^2)}}) \tag{2.2}$$

where v_0 and σ_0^2 were the value and standard deviation of the best move while v_i and σ_i^2 were the value and standard deviation of the considered move. The researcher also compared several backup values and concluded that mix operator was overall the best, while mean operator was more accurate when the number of simulations were low and max operator was more accurate when the number of simulations were high.

Kocsis and Szepesvari made significant improvement to MCTS by applying a bandit algorithm, UCB1 on Monte Carlo planning resulting in an algorithm called UCT [108]. UCB1 was initially used in multi armed bandit problem to find the right balance between exploitation and exploration. Exploration of

suboptimal looking alternatives was essential so that no good alternatives are missed due to early estimation errors. The algorithm was shown to be consistent where the probability of selecting optimal action converges to 1 when sampling grows to infinity.

$$v_i + C\sqrt{\frac{\ln n_p}{n_i}} \tag{2.3}$$

MCTS has been successfully applied mainly in the games domain especially Go. Programs based on MCTS are now competitive with the best human players which was impossible before [83, 82]. The program Zen was able to beat a professional Go player with a handicap of four stones [116]. Google Deepmind combined MCTS with deep learning in the program AlphaGo and broke the four stone handicap barrier [162]. In addition, AlphaGo (distributed version) won all the 495 games when playing with contemporary computer Go programs like Crazy Stone, Zen, Pachi, Fuego, and GnuGo, with no handicaps. Go is difficult for a computer as it has large branching factor, deep tree and evaluation function is inefficient, even if available. Breakthrough achievements in Go have sparked interests among AI Researchers and they are expecting to see more application of MCTS in various domains.

While MCTS methods are popular in games decision making, it is rarely used for combinatorial optimization problems. Runarsson et al. compared greedy, pilot and MCTS algorithms on Job Shop Scheduling [154]. In pilot setting, the tree was traversed in an exploratory only strategy and rollout was executed deterministically based on heuristics. In MCTS, a single tree exploration was carried out where tree traversal was based on e-greedy while simulation was purely random. The author reported that MCTS outperformed the pilot method on small instances, judged by the number of optimal solutions and averages. MCTS was comparable to pilot method on medium instances. In addition, MCTS managed to find optimal solutions on medium instances which were never found by the pilot method. However, the pilot method outperformed MCTS on large instances. Computation times were higher in pilot compared to MCTS for the same rollout count.

Schadd et al. proposed an adapted MCTS called Single Player Monte Carlo Search (SP-MCTS) [158]. SameGame, a one player game (puzzle) was used as

test bed as no good evaluation function is available and it fits the specialty of MCTS which does not require an evaluation function. The author performed one large search from an initial position to the end. The author added a third term to the original selection strategy of UCT where moves were chosen maximizing the formula.

$$\overline{X} + C\sqrt{\frac{\ln t(N)}{t(N_i)}} + \sqrt{\frac{\sum x^2 - t(N_i) \cdot \overline{X}^2 + D}{t(N_i)}} \tag{2.4}$$

where $t(N)$ is a visit count of node N, $t(N_i)$ is a visit count of child node N_i, $\overline{X}$ is the average value, $\sum x^2$ is sum of squared results stored so far, C is an exploration constant, D is to ensure rarely explored nodes are considered uncertain. One node (first encountered position) was added to the tree per simulation. Simulation was based on heuristic knowledge. Average was used as the back propagation strategy. The authors highlighted the importance of having a deep search tree. The authors also claimed that exploitation is good when searching with limited time while balancing exploitation and exploration is beneficial for longer searches. They also tested meta search (where the search is restarted with a different random seed) and observed that doing several small searches produced better results than one large search.

Matsumoto et al. applied SP-MCTS on reentrant scheduling problems [126]. Strategies for selection, simulation, expansion and backpropagation were identical. Specific heuristics were used for the problem. The proposed method obtained better results than the usual methods for all the cases tested.

Chaslot et al. applied MCTS on production management problems [49]. The author compared the method with fixed planning heuristics (FPH) and evolutionary planning heuristics (EPH) where heuristics were generated automatically by a neural network. MCTS performed better and faster on larger problems. However, it did not fare well on small problems. A few suggestions for improvement were recommended including playing pseudo-random moves instead of full random moves.

The application of MCTS on variants of the traveling salesman problems can be found in [140, 139, 144, 148].

2.8 Hybrid Algorithms

Hybridization has led to good quality results in previous research (e.g. [146, 19]). More recent work has validated these earlier findings. For example, [92] hybridized mixed integer linear programming, a greedy heuristic, two local search strategies and three meta-heuristics for a vehicle routing problem, reporting positive results.

GA is further extended and combined with local search to become the so called memetic algorithm. The term memetic was coined by Moscato [133, 132]. The synergy is effective as it combines the explorative capability of evolutionary algorithms and the exploitive capability of local search. Burke and Silva presented the design of memetic algorithms for scheduling and timetabling problems [39]. Zeb et al. [179] hybridized simulated annealing (SA) and a GA. The GA was used as an exploration operator, SA was used to intensify the search. The algorithm was evaluated on 35 cell formulation benchmark instances, producing 24 best results, and two new best results. A genetic algorithm was also used in [122]. Their abstract states "*Meta-heuristics still suffers from several problems that remains open as the variability of their performance depending on the problem or instance being solved. One of the approaches to deal with these problems is the hybridization of techniques.*" They concluded that their hybridized approach is the best performing method for instances with high dimensionality.

Ant based implementations are usually hybridized with local search. At each iteration, events are assigned based on pheromone information. The constructed solutions are further improved by using a local search procedure (ejection chain, SA) before the pheromone information is updated. In fact, the local search component is acknowledged to play an important role in the good results reported in [164, 165, 135].

Some authors combined several algorithms into one by passing around the mode of execution in a cyclic manner when certain bounds or idle iterations are reached. The authors hope that the algorithms will assist one another when the search is trapped in a local optima. Examples of this type of hybrid algorithms can be found in [134, 161].

Tabu Search is also often hybridized with other local search methods. A tabu list is used to keep track of the recently visited solutions. Usually, only moves that are not tabu are accepted. The accepted move is then added to the list and prohibited for some time. As a result, revisiting of the same solution is reduced and the search is able to explore the search space better. Interested readers can refer to [17, 55, 15]. Some authors also used a tabu list to keep under performing neighborhood structures or heuristics from being selected. Examples can be found in [174, 35].

2.9 Conclusion

We have reviewed the scientific literature on educational timetabling where key introduction and surveys are provided. There are three variants of educational timetabling identified, namely school, course and examination timetabling. We described how timetabling is closely associated with the graph colouring problem. The algorithms used to solve timetabling problems can be classified into one-stage, multi-stage or multi-stage with relaxation. One the most popular approaches utilized in educational timetabling problems are meta-heuristics which can be categorized as local search based (SD, FD, TS, GD, SA, ILS, VNS) and evolutionary based (GA, ACO). Another popular approach is hyper-heuristics which aim to be generally applicable across optimization problems. In addition, we presented a review of MCTS for combinatorial optimization problems. As far as we are aware, MCTS has never been utilized for any timetabling problems. Finally, we also reviewed the use of hybrid algorithms.

Chapter 3

Literature Review: Course Timetabling

3.1 Problem Description

There are many variants of the course timetabling problem, with different requirements expressed as either hard or soft constraints, across institutions of higher learning around the globe. Different implementations have reported varying degrees of success. However, it is difficult to compare the effectiveness of different algorithms if they are executed on different problem instances. Researchers have shared datasets so that algorithm comparison is more objective. The datasets utilized in this research are publicly available and regarded as the standard benchmarks:

- **Socha (11 instances)[1].** The instances (5 small, 5 medium and 1 large) are generated using an algorithm developed by Ben Paechter. In fact, this benchmark is named after the author. The time limit for the small, medium, and large instances is set to 90, 900, 9000 seconds respectively [164]. This is potentially problematical as different machine specifications means that running for 900 seconds is not a fair comparison. Refer to Table 3.1 for the benchmark statistics.

- **International Timetabling Competition 2002 (ITC02) (20 in-**

[1] http://iridia.ulb.ac.be/supp/IridiaSupp2002-001/index.html Last accessed: June 13, 2017.

stances)[2]**.** This competition was organized by the Metaheuristic Network and the instances were generated by Ben Paechter. The time limit is benchmarked by running a program on the host machine, which enables a fair comparison. Refer to Table 3.2 for the benchmark statistics.

- **International Timetabling Competition 2007 (ITC07) (24 instances)**[3]**.** The time limit is benchmarked in the same way as ITC02. Refer to Table 3.3 for the benchmark statistics. The details of the competition are given in [128].

- **Hard (60 instances).** The instances (20 small, 20 medium, 20 big) were created by Lewis and Paechter and were intended to be harder in terms of feasibility than those used in ITC02. Lewis and Paechter claimed that existing sequential constructive algorithms were only able to schedule around 60-80% of the events [120]. The time limit was set to 30, 200, 800s for the small, medium and big instances respectively. Refer to Table 3.4, 3.5 and 3.6 for statistics.

Instance	S	M	L
# of Events	100	400	400
# of Rooms	5	10	10
# of Features	5	5	10
# of Students	80	200	400

Table 3.1: Statistics for the Socha instances

Instance	1	2	3	4	5	6	7	8	9	10
# of Events	400	400	400	400	350	350	350	400	440	400
# of Rooms	10	10	10	10	10	10	10	10	11	10
# of Features	10	10	10	5	10	5	5	5	6	5
# of Students	200	200	200	300	300	300	350	250	220	200
Instance	11	12	13	14	15	16	17	18	19	20
# of Events	400	400	400	350	350	440	350	400	400	350
# of Rooms	10	10	10	10	10	11	10	10	10	10
# of Features	6	5	6	5	10	6	10	10	5	5
# of Students	220	200	250	350	300	220	300	200	300	300

Table 3.2: Statistics for the ITC02 instances

[2]http://www.idsia.ch/Files/ttcomp2002/ Last accessed: June 13, 2017.
[3]http://www.cs.qub.ac.uk/itc2007/ Last accessed: June 13, 2017.

Instance	1	2	3	4	5	6	7	8
# of Events	400	400	200	200	400	400	200	200
# of Rooms	10	10	20	20	20	20	20	20
# of Features	10	10	10	10	20	20	20	20
# of Students	500	500	1000	1000	300	300	500	500
Instance	9	10	11	12	13	14	15	16
# of Events	400	400	200	200	400	400	200	200
# of Rooms	10	10	10	10	20	20	10	10
# of Features	20	20	10	10	10	10	20	20
# of Students	500	500	1000	1000	300	300	500	500
Instance	17	18	19	20	21	22	23	24
# of Events	100	200	300	400	500	600	400	400
# of Rooms	10	10	10	10	20	20	20	20
# of Features	10	10	10	10	20	20	30	30
# of Students	500	500	1000	1000	300	500	1000	1000

Table 3.3: Statistics for the ITC07 instances

Instance	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
# of Events	200	210	200	200	200	200	200	225	225	220
# of rooms	5	6	6	5	5	5	5	5	5	5
# of Features	5	5	5	8	8	3	3	10	10	10
# of Students	200	400	400	500	500	1000	800	1000	900	1000
Instance	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20
# of Events	200	225	225	225	200	200	200	225	225	225
# of Rooms	5	5	5	5	5	5	5	5	5	5
# of Features	4	10	10	3	3	3	3	3	3	3
# of Students	1000	1000	1000	1000	900	900	900	1000	1000	1000

Table 3.4: Statistics for the Hard small instances

Instance	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10
# of Events	400	390	390	410	410	410	410	400	400	400
# of Rooms	10	10	10	10	10	11	11	10	10	10
# of Features	10	10	10	9	9	10	10	10	10	8
# of Students	400	400	400	400	450	450	450	400	400	500
Instance	M11	M12	M13	M14	M15	M16	M17	M18	M19	M20
# of Events	400	400	400	400	425	400	400	400	410	410
# of Rooms	8	8	8	8	8	8	8	8	8	8
# of Features	6	5	6	5	10	6	10	10	5	5
# of Students	800	800	800	1000	500	1000	800	1000	1000	1000

Table 3.5: Statistics for the Hard medium instances

Instance	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10
# of Events	1000	1000	1000	1050	1075	1075	1050	1025	1050	1075
# of Rooms	28	25	25	25	25	25	25	25	25	25
# of Features	20	20	20	20	20	20	20	20	20	20
# of Students	1000	1000	900	800	1000	1000	1100	1000	800	1000
Instance	B11	B12	B13	B14	B15	B16	B17	B18	B19	B20
# of Events	1075	1000	1000	1000	1000	1000	1000	1000	1000	1000
# of Rooms	25	26	25	25	25	25	25	25	25	25
# of Features	20	25	25	25	25	10	10	10	10	10
# of Students	1000	1000	1000	1000	1000	1000	1200	1000	1000	1000

Table 3.6: Statistics for the Hard big instances

Solving the course problem involves assigning a set of C courses (with a set of F features and attended by S students) to 45 time slots (5 days of 9 hours each) and a set of R rooms. The objective is to satisfy all hard constraints and minimize soft constraint violations as far as possible. Perfect solutions are known to exist for all the datasets (except Hard). A sample course timetable depicting the 45 time slots with R rooms each, is shown in Fig. 3.1.

	8-9am				9-10am	10-11am	11-12pm	12-1pm	1-2pm	2-3pm	3-4pm	4-5pm			
Mon	1				2	3	4	5	6	7	8	9			
	r_1	r_2	...	$r_{\|R\|}$	...	...	...	...	...	...	...	r_1	r_2	...	$r_{\|R\|}$
Tue	10				11	12	13	14	15	16	17	18			
	r_1	r_2	...	$r_{\|R\|}$	...	...	...	...	...	...	...	r_1	r_2	...	$r_{\|R\|}$
Wed	19				20	21	22	23	24	25	26	27			
	r_1	r_2	...	$r_{\|R\|}$	...	...	...	...	...	...	...	r_1	r_2	...	$r_{\|R\|}$
Thu	28				29	30	31	32	33	34	35	36			
	r_1	r_2	...	$r_{\|R\|}$	...	...	...	...	...	...	...	r_1	r_2	...	$r_{\|R\|}$
Fri	37				38	39	40	41	42	43	44	45			
	r_1	r_2	...	$r_{\|R\|}$	...	...	...	...	...	...	...	r_1	r_2	...	$r_{\|R\|}$

Figure 3.1: A sample course timetable.

The hard constraints for all the datasets are:

- HC1: No student can be assigned more than one course at the same time.
- HC2: The room should satisfy the features required by the course.
- HC3: The number of students attending the course should be less than or equal to the capacity of the room.
- HC4: No more than one course is allowed for each time slot in each room.

There are two additional hard constraints for ITC07 namely:

- HC5: A course can only be assigned to some preset time slots
- HC6: Where specified, a course should be scheduled to occur in the correct order.

The soft constraints for all the datasets (except Hard) are:

- SC1: A student should not have a single course on a day.
- SC2: A student should not have more than two consecutive courses.
- SC3: A student should not have a course scheduled in the last time slot of the day.

3.2 Formal Representation of The Problem

The course timetabling problem can be formulated as (adapted from [169]):

$$Minimize \sum_{i=1}^{3} SC_i \tag{3.1}$$

SC_1: Penalty for students with one event on a day

$$\sum_{s \in S} \sum_{d=1}^{5} z_{s,d} \tag{3.2}$$

SC_2: Penalty for students with three or more events consecutively.

$$\sum_{s \in S} \sum_{d=1}^{5} \sum_{t=(d-1)\times 9+1}^{d\times 9-2} y_{s,t} \cdot y_{s,(t+1)} \cdot y_{s,(t+2)} \tag{3.3}$$

SC_3: Penalty for students with one event in the last time slot of the day

$$\sum_{s \in S} \sum_{t \in \{9,18,\ldots 45\}} y_{s,t} \tag{3.4}$$

subject to:

HC1: No student is required to attend more than one event at the same time.

$$\sum_{r \in R} x_{etr} \cdot a_{se} \leq 1 \qquad e \in E, s \in S, t \in T \tag{3.5}$$

HC2: Each event is assigned a room with enough seats for all attending students and all features required.

$$b_{er} \cdot c_{er} \cdot x_{etr} = x_{etr} \qquad e \in E, r \in R, t \in T \tag{3.6}$$

HC3: Only one event per room in any time slot.

$$\sum_{e \in E} x_{etr} \leq 1 \qquad r \in R, t \in T \tag{3.7}$$

HC4: Events are assigned to designated time slots.

$$i_{et} \cdot x_{etr} = x_{etr} \qquad e \in E, r \in R, t \in T \tag{3.8}$$

HC5: Where specified, events should be scheduled in the correct order.

$$j_{e,t_m} \cdot k_{e,t_m} \cdot x_{et_m r} = x_{et_m r} \qquad e \in E, r \in R, t_m \in T \tag{3.9}$$

where:

set of events, $E = \{e_1, \ldots, e_{|E|}\}$
set of time slots, $T = \{1, \ldots, 45\}$
set of rooms, $R = \{r_1, \ldots, r_{|R|}\}$
set of students, $S = \{s_1, \ldots, s_{|S|}\}$
set of features $F = \{f_1, \ldots, f_{|F|}\}$
set of days, $D = \{1, \ldots, 5\}$
set of events that must appear later than e, A_e
set of events that must appear earlier than e, B_e

$$a_{s,e} = \begin{cases} 1 & \text{if student } s \text{ attends event } e \\ 0 & \text{otherwise} \end{cases} \tag{3.10}$$

$$b_{e,r} = \begin{cases} 1 & \text{if size of event } e \leq \text{capacity of room } r \\ 0 & \text{otherwise} \end{cases} \tag{3.11}$$

$$c_{e,r} = \begin{cases} 1 & \text{if } \sum\limits_{f \in F} g_{e,f} \cdot h_{r,f} = \sum\limits_{f \in F} g_{e,f} \\ 0 & \text{otherwise} \end{cases} \tag{3.12}$$

$$g_{e,f} = \begin{cases} 1 & \text{if event } e \text{ requires feature } f \\ 0 & \text{otherwise} \end{cases} \tag{3.13}$$

$$h_{r,f} = \begin{cases} 1 & \text{if room } r \text{ has feature } f \\ 0 & \text{otherwise} \end{cases} \tag{3.14}$$

$$i_{e,t} = \begin{cases} 1 & \text{if event } e \text{ can be assigned to time slot } t \\ 0 & \text{otherwise} \end{cases} \tag{3.15}$$

$$j_{e,t_m} = \begin{cases} 1 & \text{if } \sum_{e_v \in A_e} x_{e_v t_n r} \cdot p_{t_m,t_n} = \sum_{e_v \in A_e} x_{e_v t_n r} \\ 0 & \text{otherwise} \end{cases} \tag{3.16}$$

$$k_{e,t_m} = \begin{cases} 1 & \text{if } \sum_{e_v \in B_e} x_{e_v t_n r} \cdot q_{t_m,t_n} = \sum_{e_v \in B_e} x_{e_v t_n r} \\ 0 & \text{otherwise} \end{cases} \tag{3.17}$$

$$p_{t_m,t_n} = \begin{cases} 1 & \text{if } t_m < t_n \\ 0 & \text{otherwise} \end{cases} \tag{3.18}$$

$$q_{t_m,t_n} = \begin{cases} 1 & \text{if } t_m > t_n \\ 0 & \text{otherwise} \end{cases} \tag{3.19}$$

$$x_{e,t,r} = \begin{cases} 1 & \text{if event } e \text{ is assigned to time slot } t \text{ and room } r \\ 0 & \text{otherwise} \end{cases} \tag{3.20}$$

$$y_{s,t} = \begin{cases} 1 & \text{if } x_{etr} \cdot a_{se} = 1 \quad r \in R, t \in T \\ 0 & \text{otherwise} \end{cases} \tag{3.21}$$

$$z_{s,d} = \begin{cases} 1 & \text{if } \sum_{t=(d-1)\times 9+1}^{d\times 9} y_{s,t} = 1 \quad d \in D, s \in S \\ 0 & \text{otherwise} \end{cases} \tag{3.22}$$

3.3 Related Work

3.3.1 Specific approaches applied to Socha instances

One of the earliest approaches on these instances was based on an ant system [164]. Ants follow a list of ordered events and choose time slots randomly based on probabilities that depend on pheromone and heuristic information. A matching algorithm was applied for room assignment. The candidate solution was further improved by a local search as an exploitation mechanism. A global best solution was maintained. Pheromone corresponding to the global best was increased, while other pheromone trails were reduced using an evaporation coefficient. The pheromone levels were reduced to allow exploration in the search space. Parameter tuning was required for each instance type. The method was reported to be better than random restart local search. Interested readers can

refer to its variants [71] [102].

Burke et al. [35] proposed a method called Tabu Search Hyper-Heuristic to overcome the weaknesses of optimization methods specifically meta-heuristics, which often require intensive parameter tuning for individual instances. They aimed to develop a general approach which can be easily applied to different problems, yet remains competitive with state of the art approaches. The method selects heuristics at each decision point instead of solving the problem directly. Heuristics were ranked according to their performance inspired by the principles of reinforcement learning. The value of the selected heuristic is increased by 1 when applied and which results in an improvement to the current cost function. Otherwise, it is decreased by 1. A tabu list was also implemented to restrict the use of heuristics which did not recently perform well based on First-In, First-Out (FIFO). A heuristic placed in the list is made tabu even if it has the highest rank. The approach was competitive with ant systems and random restart local search.

Abdullah et al. [3] used a saturation degree heuristic to get an initial feasible solution where soft constraint violations were ignored. The solution was then improved using Variable Neighbourhood Search (VNS). VNS was made up of three stages namely shaking, local search and move. Variants tested were VNS-basic which only accepted improved solutions and VNS-EMC which employed an exponential monte carlo acceptance criterion where worse solutions were accepted with a certain probability. VNS-EMC performed better than VNS-basic and the authors believed that the latter got trapped in a local optima. The result was further improved when a tabu list was utilized on VNS-EMC to prevent neighborhoods that had not performed well recently from being chosen in the next evaluation. The author also showed that VNS with ordering performed better than VNS without ordering. It was able to get optimal solutions for all the small instances. However, the constructive heuristic failed to get feasible solution for the large instance.

Abdullah et al. [5] proposed Randomized Iterative Improvement. At each iteration, the best solution was selected among temporary solutions produced from different neighborhood structures where improved solutions compared to the best were always accepted, while worse solutions were accepted based on an exponential monte carlo acceptance criterion. The authors also introduced

three more neighbourhood structures. The methodology was able to produce better or equal results on seven instances. It failed to get feasible solutions for the large instance.

Abdullah et al. employed a Memetic Algorithm which combined a Genetic Algorithm with local search [4]. Randomized Iterative Improvement was implemented as the local search after a mutation phase. A crossover operator was not used which avoided the need to repair the solution. The methodology performed better with reduced complexity compared to when crossover was used [8]. Improved results were reported compared to Randomized Iterative Improvement alone. The method was able to produce feasible solutions for the large instance by using constructive heuristics which added and removed events.

Landa Silva and Obit modified the original Great Deluge by replacing the linear decay rate with a non linear function [113]. The water level was allowed to go up when it was about to converge with the candidate solution. The rationale was to accept worse solutions in order to better explore the search space. The method required additional parameter tuning.

Turabieh and Abdullah incorporated Tabu Search into a memetic approach [174]. A tabu list was maintained in a First-In, First-Out (FIFO) manner to hold ineffective neighborhood structures from being selected in the next iteration. A good neighborhood structure was used in the next iterations until no better solutions were obtained. Good results were reported.

Landa Silva and Obit tested Non-Linear Great Deluge with Genetic Algorithms without a crossover operator [114]. Local search was applied after mutation phase. Better result was reported compared to Non-Linear Great Deluge alone.

Abdullah et al. tackled the problem with Great Deluge and Tabu Search [7]. Least saturation degree was used to produce feasible solutions. If a feasible solution could not be found, neighborhood moves were applied. In the improvement phase, during each iteration, the best solution among solutions generated from Great Deluge and Tabu Search was accepted only if it was better than the best. The water level in the Great Deluge was allowed to increase when a non-improving counter reached a certain level to allow flexibility in accepting

worse solutions.

Obit et al. utilized Non-Linear Great Deluge with reinforcement learning [136] on the instances. Heuristics or neighborhood structures, were selected probabilistically based on their weights, instead of choosing randomly. The weights were increased or decreased based on their performance. Two types of Modified Choice Function (MCF) are investigated namely MCF with static memory and MCF with random learning rate. For MCF with static memory, a reward of 1 point is awarded to the chosen heuristic if the current solution is improved, otherwise no point is awarded. The weights are updated in every predefined period. For MCF with random learning rate, a different set of rewards was used according to the difference between the best cost and the current cost. In addition, the reward was weighted by a random value in the range (0.5, 1.0]. The method involved the acceptance and rejection of solutions using Non-Linear Great Deluge acceptance criterion.

Turabieh et al. presented a Fish Swarm Intelligent Algorithm [176]. This work simulated fish movements when searching for food. At the initialization phase, constructive heuristics were used to generate feasible solutions. In the improvement phase, *visual scope* was defined. If the *visual scope* was empty, Steepest Descent was applied. If the *visual scope* was crowded, Great Deluge was applied (with Nelder-Mead Simplex algorithm to intelligently adjust the decay rate). The authors believed by accepting worse solutions, the algorithm was able to escape from local optimum. If *visual scope* was not crowded, Great Deluge with best solution or central point as estimated quality was applied. The method was able to produce best known results at that time.

Shaker and Abdullah tried to control multiple algorithms in a round robin fashion [161]. The algorithms were Hill Climbing, Great Deluge and Simulated Annealing. Each algorithm was assigned an equal portion of time and utilized in a First-In, First-Out (FIFO) manner.

Sabar et al. used Honey Bee Mating to tackle the problem [155]. The authors employed hybrid graph colouring heuristics to generate an initial feasible solution. Events were sorted (by least saturation degree, largest degree and largest enrollment) and assigned to time slots and rooms randomly. There was no guarantee of generating a feasible solution. Next, the best solution from the

pool was set as the queen (best solution). During mating, drones (incumbent solutions) were selected, where the fitter ones had a better chance of being selected. During breeding, broods (new solutions) were generated by a crossover operator that took place between queen and selected drones. After breeding, simple descent was employed to improve the broods. If any of the improved brood was better than the queen, the queen was replaced. All the drones were destroyed. The brood was mutated using a Kempe chain, ready for the next mating. Highly competitive results were reported. However, the author mentioned the drawback of the method was the number of parameters that need to be set.

Ceschia et al. applied simulated annealing on the problem and achieved breakthrough results in very short time relative to methods used by other researchers [48]. Two neighborhood structures were used; moving an event from one slot to another and swapping events. Dummy time slots and dummy rooms were used. The cost function was evaluated based on unscheduled events, precedences and conflicts in addition to soft constraint violations which prompted the need to set the proper weights for each component. In addition, parameters specific to simulated annealing had to be set. The author used an F-race mechanism to tune the related parameters. The author attributed the good results to the preprocessing and constraint reformulation step which improved the efficacy of the local search. Their implementation produced the best results in terms of best and mean results as reported in the literature.

Abuhamdah et al. proposed population based local search [11]. The authors believed that population based algorithms were better at exploring a search space compared to local search while local search was better at exploiting the search space. Initial feasible solutions was generated in three phases. Events were sorted by largest degree and assigned to timeslot and room randomly. If no feasible room was available, any room would be assigned. If feasibility could not be achieved, phase 2 that employed Hill Climbing, and phase 3, that employed Tabu Search, will be invoked. MPCA-ARDA was used as the local search.

Table 3.7 shows the performance comparison among the state of the art methods on Socha instances in terms of soft constraint violations. Meanwhile, the details of the solvers are given in Table 3.8. The highly tuned Simulated Annealing employed by Ceshia et al. outperformed the rest of the methods.

	Instance										
Solver	S1	S2	S3	S4	S5	M1	M2	M3	M4	M5	L
A1	1	3	1	1	**0**	195	184	248	164.5	219.5	851.5
A2	1	2	**0**	1	**0**	146	173	267	169	303	1166
A3	10	9	7	17	7	243	325	249	285	132	1138
A4	**0**	**0**	**0**	**0**	**0**	317	313	357	247	292	926
A5	5	5	3	3	**0**	176	154	191	148	166	798
A6	**0**(0.8)	**0**(2.0)	**0**(1.3)	**0**(1.0)	**0**(0.2)	80(101.4)	105(116.9)	139(162.1)	88(108.8)	88(119.7)	730(834.1)
A7	**0**(**0.0**)	**0**(**0.0**)	**0**(**0.0**)	**0**(**0.0**)	**0**(**0.0**)	221(224.8)	147(150.6)	246(252.0)	165(167.8)	130(135.4)	529(552.4)
A8	**0**	**0**	**0**	**0**	**0**	242	161	265	181	151	-
A9	2	4	2	**0**	4	254	258	251	321	276	1027
A10	3	4	6	6	**0**	140	130	189	112	141	876
A11	**0**	**0**	**0**	**0**	**0**	55	70	102	32	61	653
A12	**0**	**0**	**0**	**0**	**0**	78	92	135	75	68	556
A13	**0**	1	**0**	**0**	**0**	126	123	185	116	129	821
A14	**0**	**0**	**0**	**0**	**0**	38	37	60	39	55	638
A15	**0**	**0**	**0**	**0**	**0**	175	197	216	149	190	912
A16	**0**	**0**	**0**	**0**	**0**	45	40	61	35	49	407
A17	**0**	**0**	**0**	**0**	**0**	190	223	259	127	132	869
A18	**0**	**0**	**0**	**0**	**0**	117	108	135	75	160	589
A19	**0**	**0**	**0**	**0**	**0**	168	160	176	144	71	417
A20	**0**	**0**	**0**	**0**	**0**	75	88	129	74	64	523
A21	**0**(**0.0**)	**0**(**0.0**)	**0**(**0.0**)	**0**(0.1)	**0**(**0.0**)	**9**(**26.5**)	**15**(**25.9**)	**36**(**49.0**)	**12**(**23.8**)	**3**(**10.9**)	**208**(**260.0**)
A22	**0**	**0**	**0**	**0**	**0**	41	39	60	39	55	463

Table 3.7: Comparison among the state of the art methods on Socha instances. Depicted is best(mean) of soft constraint violations. Note that some authors only reported their best results.

Solver	Technique	Reference
A1	Ant System	Socha et al. [164]
A2	Tabu Search Hyperheuristic	Burke et al. [35]
A3	Fuzzy Multiple Heuristic	Asmuni et al. [16]
A4	Variable Neighbourhood Search	Abdullah et al. [3]
A5	Co-operative Ant	Ejaz and Javed [71]
A6	Extended Great Deluge	McMullan [129]
A7	Memetic (Genetic + Local Search)	Abdullah et al. [4]
A8	Randomized Iterative Improvement	Abdullah et al. [5]
A9	Genetic Algorithms + Local Search	Abdullah et al. [8]
A10	Non-Linear Great Deluge	Landa Silva and Obit [113]
A11	Tabu Search + Memetic	Turabieh and Abdullah [174]
A12	Great Deluge + Tabu Search	Abdullah et al. [7]
A13	Evolutionary Great Deluge	Landa Silva and Obit [114]
A14	Non Linear Great Deluge + Learning	Obit et al. [136]
A15	Electromagnetism + Great Deluge	Turabieh et al. [175]
A16	Fish Swarm	Turabieh et al. [176]
A17	Elitist Ant System	Jaradat and Ayob [102]
A18	Round Robin Multi Algorithms	Shaker and Abdullah [161]
A19	Modified Harmony Search	Al-Betar and Khader [12]
A20	Honey Bee Mating	Sabar et al. [155]
A21	Simulated Annealing	Ceschia et al. [48]
A22	Population Based Local Search	Abuhamdah et al. [11]

Table 3.8: Details of solvers applied on Socha instances.

3.3.2 Specific approaches applied to ITC02 instances

Kostuch employed a simulated annealing based heuristic approach, becoming the winner of International Timetabling Competition 2002 (ITC2002) [109]. In the preprocessing, the author defined the event room matrix and incidence matrix. The incidence matrix was further updated with 1-room events. In finding an initial feasible solution, events were ordered and each assigned a time slot with a minimum number of events, provided that the number of events in the time slot did not exceed the number of rooms. Unassigned events were placed in a pool. Maximum matching was run for room assignment and unassigned events were removed from the time slots and placed in a pool. Next, in an *improvement* phase, unplaced events were refitted into the time slots where events were removed during the room assignment phase. In a *shuffling* phase, every event from the pool of unplaced events was assigned to a random time slot and maximum matching was run for room assignment. The newly unassigned event was hopefully different from the initially unassigned event. The *improvement* phase

was rerun. Next in a *blow-up* phase, the unassigned events were placed into a time slot and all current events in that time slot were removed. Then rooms were assigned and unplaced events were kept in a pool. The *improvement* and *shuffling* phase were rerun. The still unplaced events, if there were any, were distributed over the last time slots. The feasible solution was then improved with simulated annealing by sequencing the time slots and exchanging pairs of events. Finally, simulated annealing was run with lower acceptance probability on the best solution until the time limit was reached. The search was confined to the vicinity of solution.

Cordeau et al. used Tabu Search on ITC02 and ranked second for the competition [55]. They first found a feasible solution and then applied Tabu Search using an evaluation function defined as the weighted sum of hard and soft constraints. The search was allowed to navigate into infeasible regions where a parameter was varied at each iteration to control the feasibility of the solution. A few perturbation procedures as well as ejection chain were also used.

Burke et al. proposed a Great Deluge approach to the problem [27]. The authors raised doubt as to the practicality of using Simulated Annealing in solving problems as it requires significant time and experience from a user in setting parameters. The method extended the Great Deluge by Dueck (which accepts every solution whose objective function is less than or equal to an upper level) by accepting all candidate solutions which were better than the current one. The one parameter which had to be set was the decay rate as a function of search time and expected search quality. The expected search quality had to be approximated by employing a Hill Climbing algorithm. The authors claimed that the method produced better results for longer searches. The method was superior in terms of performance from comparisons made with other methods such as Simulated Annealing, Threshold Algorithm and Hill Climbing. The author suggested good initial solutions, non-linear level function, hybridization and neighborhood structures as future research directions.

Arntzen and Lokketangen employed a Tabu Search heuristic for ITC02 instances [15]. In the constructive solver, at each iteration, an event with the fewest possible places was selected and assigned to a place which was selected by minimizing a weighted sum. Once a feasible solution was found, the basic search was started. At each iteration, a list of moves were sorted by value and

frequency. The best move in the list was chosen if it resulted in a solution which was better than the best solution. Otherwise, the best non-tabu move was chosen with 50% probability. If it was not chosen, the next best non-tabu move was chosen with 50% probability. The move selection was repeated until a move was chosen. The moved event was marked as tabu according to certain tenures. Ejection chain was called after 4000 moves without improvement to the best value. A tabu mechanism was also used in the ejection chain where an event was set tabu for a certain number of steps when it was moved. Basic search and ejection chain were alternated until the time limit was reached.

Chiarandini et al. employed a strategy which combined construction heuristics, variable neighborhood descent and simulated annealing which outperformed the winner of ITC2002 [52]. The authors used a racing algorithm to iteratively select and configure algorithms. Candidate algorithms were evaluated and discarded when sufficient statistical evidence was gathered against them. Local search and tabu search were utilized to obtain a feasible solution. The feasible solution is improved in terms of soft constraint violations by using variable neighborhood descent and simulated annealing. The authors claimed that the method reduced the number of experiments and was well suited for the engineering of meta-heuristics. Findings highlighted the importance of local search in ant colony optimization and genetic algorithms and that variable neighborhoods strongly enhanced the local search. Solving hard and soft constraints separately was also found to be preferable than weighting constraints in an evaluation function. The authors also highlighted that tabu search was not suitable for optimizing soft constraints. Population based meta-heuristics did not perform better than a single solution based approach and the importance of problem specific knowledge was emphasized. The method obtained better results than the ITC2002 winner on 18 out of 20 instances.

Kostuch further improved his method and achieved the best results on all 20 instances [110]. Feasible solutions were constructed using graph colouring and maximum matching. The feasible solution was improved by sequencing the time slots and exchanging pairs of events. To keep the neighborhood structure simple, the author also introduced 10 dummy events, 2 at each end of day time slots which were removed in the final timetable. His implementation is the current state of the art for the problem instances.

Ceschia et al. tested their variant of Simulated Annealing on the problem as well, using F-race [48]. However, the results reported were inferior to the best in the literature.

Table 3.9 shows the performance comparison among the state of the art methods on ITC02 instances in terms of soft constraint violations. The details of the solvers are given in Table 3.10. No other solvers have yet to beat the results of the Simulated Annealing implementation by Kostuch in any of the instances either in terms of best or mean.

	Instance									
Solver	1	2	3	4	5	6	7	8	9	10
B1	45	25	65	115	102	13	44	29	17	61
B2	61	39	77	160	161	42	52	54	50	72
B3	85	42	84	119	77	6	12	32	184	90
B4	63	46	96	166	203	92	118	66	51	81
B5	132	92	170	265	257	133	177	134	139	148
B6	45	14	45	71	59	1	3	1	8	52
B7	**16**(**30.2**)	**2**(**11.4**)	**17**(**31.0**)	**34**(**60.8**)	**42**(**72.1**)	**0**(**2.4**)	**2**(**8.9**)	**0**(**2.0**)	**1**(**5.8**)	**21**(**35.0**)
B8	45(57.1)	20(33.2)	43(53.2)	87(109.9)	71(91.7)	2(14.1)	**2**(13.7)	9(20.0)	15(21.9)	41(60.7)

	Instance									
Solver	11	12	13	14	15	16	17	18	19	20
B1	44	107	78	52	24	22	86	31	44	7
B2	53	110	109	93	62	34	114	38	128	26
B3	73	79	91	36	27	300	79	39	86	0
B4	65	119	160	197	114	38	212	40	185	17
B5	35	290	251	230	140	114	186	87	256	94
B6	30	75	55	18	8	55	46	24	33	**0**
B7	**5**(**12.9**)	**55**(**76.3**)	**31**(**47.1**)	**11**(**22.3**)	**2**(**8.4**)	**0**(**3.4**)	**37**(**54.0**)	**4**(**9.4**)	**7**(**16.4**)	**0**(**0.5**)
B8	24(38.2)	62(83.7)	59(78.0)	21(34.2)	6(11.8)	6(16.7)	42(56.5)	11(25.9)	56(73.0)	**0**(1.8)

Table 3.9: Comparison among the state of the art methods on ITC02 instances. Depicted is best(mean) of soft constraint violations. Note that some authors only reported their best results.

Solver	Technique	Reference
B1	Simulated Annealing	Kostuch [109]
B2	Tabu Search	Cordeau et al. [55]
B3	Great Deluge	Burke et al. [27]
B4	Local Search + Tabu Search	DiGaspero and Schaerf [63]
B5	Local Search Heuristic	Arntzen and Lokketangen [15]
B6	Hybrid Algorithm	Chiarandini et al. [52]
B7	Simulated Annealing	Kostuch [110]
B8	Simulated Annealing	Ceschia et al. [48]

Table 3.10: Details of solvers applied on ITC02 instances.

3.3.3 Specific approaches applied to ITC07 instances

The submission by Cambazard et al. won the post enrolment based course timetabling competition, ITC2007 [41]. A few approaches were studied. In the first approach, local search is performed on randomly generated solutions to find a feasible one. A tabu list is maintained to prevent an event from being assigned the same time slots for the last k iterations. Among the neighborhood structures used were; moving an event to an empty space, swapping two events, swapping two time slots, matching where events are reassigned within a time slot and moving an event with matching and Hungarian move. The feasible solution is optimized by simulated annealing with reheating. Moving an event with matching is the only neighborhood structure considered in this phase. The second approach presented was also based on local search but with a relaxation on room allocation. It was termed LS with colouring. Four stages were involved. In stage 1, a feasible solution is found ignoring room allocation. In stage 2, soft constraint violations were minimized, again ignoring room allocation. In stage 3, the solution is repaired into a feasible solution. In stage 4, the solution is improved in terms of soft constraint violations. The same neighborhood structures were used but without matching during room allocation. The author reported LS with colouring was the best approach in finding feasible solutions. Also LS with colouring worked best for highly constrained problems. Constraint programming was also developed but was less competitive compared to the local search approach and it was unable to find feasible solutions for instances 1,2,9 and 10.

Chiarandini et al. employed a modular multiphase heuristic solver for the post enrolment course timetabling problem of ITC07 [53]. There were two

phases; a hard constraint solver and a soft constraint minimizer. In the hard constraint solver, partial feasible solutions were generated. Lectures were ordered and assigned to a time slot that can fit the fewest unscheduled lectures. Unassigned events were kept in a list and handled by a Tabu Search based on the PARTIALCOL algorithm [22]. At each iteration, unassigned events were selected based on enrollment and placed into the best non-tabu time slot and all conflicting events (if there were any) were moved into the unassigned list. After a fixed number of non-improving iterations, the solution was perturbed by the soft constraint optimizer. The Tabu Search and soft constraint optimizer were alternated until a feasible solution was found or the time limit was reached. The soft constraint optimizer was then applied to the feasible solution. In this phase, local search was employed on several neighborhood structures such as one exchange, two exchange, time slots swap and Kempe chains until no improvement was found. Finally, Simulated Annealing was performed by moving and swapping events with matching until the end of the allowed time.

Nothegger et al. applied Ant Colony Optimization (ACO) to ITC2007 achieving fourth place in the competition [135]. They proposed two separate matrices to store pheromone information instead of the traditional single matrix. They showed that a two matrices representation produced better results in terms of distance to feasibility (DTF) and soft constraints penalty (SCP) as it is less expensive computationally, thus allowing more iterations per time unit. Events were considered in random order and assigned to time slots and rooms based on pheromone information. Heuristic information in a typical ACO was not used to promote randomness. Each constructed solution were improved locally by an ejection chain. Pheromone information is updated by solutions with lowest DTF scores and better than average SCP scores. The pheromone levels were reduced by evaporation. The authors also presented a parallel ACO with simulated annealing as the local search procedure.

Muller tested a hybrid algorithm on ITC07, being placed fifth in the competition [134]. Iterative Forward Search was used to find a complete solution, where during each iteration the worst variable (domain size to hard constraint ratio) was assigned the value that violated the least number of hard constraints and then all conflicting variables were unassigned. Furthermore, conflict based statistics was used to prevent repetitive assignments. The completed solution was further improved by executing several algorithms in a cyclic fashion when

certain bounds or idle iterations were reached. The algorithms utilized were Hill Climbing, Great Deluge and Simulated Annealing. Events were assigned to time slots without rooms. Room assignments and precedence were treated as soft constraints. The weights were initially assigned a value of 1 and increased after 1000 idle iterations. Among the neighborhoods used were time-move, room-move, event-move, event-swap and precedence-swap.

Lewis presented an algorithm based on Simulated Annealing for ITC07 [117]. The problem was tackled in three stages, each phase using one third of the time limit. Remaining time was passed to the next stage and the algorithm might also stop early if all the stages were completed. In the first stage, events were inserted into the timetable, obeying all the hard constraints except for precedence constraints. Events were chosen based on saturation degree and ties were broken randomly. Least constraining places were chosen where ties were broken by prioritizing time slot with fewest events and further ties were broken randomly. Unplaced events were kept in a list. Attempts were made to insert the unplaced events after shuffling (a random event was moved to a random unoccupied place) and extractions (two assigned events were first chosen randomly where the one with the least enrollment was moved to a list). In the second stage, precedence constraints were satisfied while not violating the previous hard constraints by using Simulated Annealing. Two distinct cells were randomly selected and the contents were swapped. The initial temperature was determined by calculating the variance by performing a sample of neighborhood moves. In the third stage, an attempt was made to satisfy the soft constraints while not violating the previous hard constraints, again by using Simulated Annealing.

Ceshia et al. also utilized a highly tuned Simulated Annealing (described earlier) on ITC07 instances reporting good results [48].

Lewis and Thompson achieved 100% feasibility on all instances of ITC2007 by using constructive heuristics and followed by their PARTIALCOL algorithm which uses a tabu mechanism for the remaining unassigned events [118]. The authors further improved the method by performing perturbations in the form of a random walk and resetting the tabu list after 5000 idle iterations. The feasible solution was improved by using simulated annealing. The initial temperature was set automatically as the standard deviation of the cost of sample moves. The cooling rate was altered during the run. The authors also used the

feasibility ratio to gauge the connectivity of the search space of various neighborhood operators on the instances. A Kempe chain operator was found to be particularly suitable for the instances. Strong results were achieved. The authors also showed that the use of dummy rooms did not improve the results.

Table 3.11 shows the performance comparison among the state of the art methods on ITC07 instances in terms of soft constraint violations. The details of the solvers are given in Table 3.12. Competitive results are shown by the Simulated Annealing methods of Ceshia et al. as well as Lewis and Thompson.

	Instance							
Solver	1	2	3	4	5	6	7	8
C1	15(547.0)	9(403.0)	174(254.0)	249(361.0)	**0**(26.0)	**0**(16.0)	1(8.0)	**0**(**0**)
C2	925	1156	179	66	52	536	7	**0**
C3	**0**(613.0)	**0**(556.0)	**110**(680.0)	53(580.0)	13(92.0)	**0**(212.0)	**0**(**4.0**)	**0**(61.0)
C4	1330	2154	205	394	**0**	13	5	**0**
C5	59(399.2)	**0**(**142.2**)	148(209.9)	25(349.6)	**0**(7.7)	**0**(**8.6**)	**0**(4.9)	**0**(1.5)
C6	**0**(**377.0**)	**0**(382.2)	122(**181.8**)	**18**(**319.4**)	**0**(**7.5**)	**0**(22.8)	**0**(5.5)	**0**(0.6)

	Instance							
Solver	9	10	11	12	13	14	15	16
C1	29(1167.0)	2(1297.0)	178(361.0)	14(380.0)	**0**(135.0)	**0**(15.0)	**0**(47.0)	1(58.0)
C2	1480	1364	166	1	360	576	**0**	**0**
C3	**0**(**202.0**)	**0**(**4.0**)	143(774.0)	**0**(538.0)	5(360.0)	**0**(41.0)	**0**(**29.0**)	**0**(101.0)
C4	1895	2440	347	453	74	2	**0**	6
C5	**0**(258.8)	3(186.4)	142(269.5)	267(400.0)	1(120.0)	**0**(3.6)	**0**(48.0)	**0**(**50.1**)
C6	**0**(514.4)	**0**(1202.4)	**48**(**202.6**)	**0**(**340.2**)	**0**(**79.0**)	**0**(**0.5**)	**0**(139.9)	**0**(105.2)

	Instance							
Solver	17	18	19	20	21	22	23	24
C1	-	-	-	-	-	-	-	-
C2	-	-	-	-	-	-	-	-
C3	-	-	-	-	-	-	-	-
C4	-	-	-	-	-	-	-	-
C5	**0**(**0.0**)	**0**(41.1)	**0**(951.5)	**543**(**700.2**)	5(35.9)	5(**19.9**)	1292(1707.7)	**0**(**105.3**)
C6	**0**(0.1)	**0**(**2.2**)	**0**(**346.1**)	557(724.5)	**1**(**32.1**)	**4**(1790.1)	**0**(**514.1**)	18(328.2)

Table 3.11: Comparison among the state of the art methods on ITC07 instances. Depicted is best(mean) of soft constraint violations. Note that some authors only reported their best results.

Solver	Technique	Reference
C1	Simulated Annealing	Cambazard et al. [41]
C2	Hybrid Algorithm	Chiarandini et al. [53]
C3	Ant Colony Optimization	Nothegger et al. [135]
C4	Hybrid Algorithm	Mueller [134]
C5	Simulated Annealing	Ceschia et al. [48]
C6	Simulated Annealing	Lewis and Thompson [118]

Table 3.12: Details of solvers applied on ITC07 instances.

3.3.4 Specific approaches applied to Hard instances

Lewis and Paechter applied Grouping Genetic Algorithms (GGA) [76, 77] to find feasible solutions for the Hard instances [120]. The university course timetabling problem (UCTP) was considered as a grouping problem where groups (time slots) were treated as the building blocks for representations and genetic operators. The authors modified the recombination operator of the standard GGA which comprises four stages, namely point selection, injection, removal of duplicates using adaptation and reconstruction. In addition to recombination, other operators used were mutation and inversion. The algorithm was further improved by investigating several fitness functions and applying local search after the mutation operator. The authors concluded that in many cases, GGA was outperfomed by a local search algorithm especially on large problem instances.

Tuga et al. employed a sequential approach (ISheuristic) consisting of Least Saturation Degree (LSD) and Largest Degree (LD) to construct the initial solution [173]. Any unassigned events were placed in artificial time slots and treated as a soft constraint violation and minimized by using Hybrid Simulated Annealing (HSA). KCHeuristic was applied and the temperature adjusted when no improvement was observed after a certain number of iterations. The neighbourhood structures used were simple, swap and kempe chain. The initial temperature was set to a sufficiently high value. The cooling equation was similar to the one used by Kostuch [110]. The number of trials for each temperarure was set to $a \cdot |events|$ where a is initially set as 10 and linearly increased. When compared to the method by Lewis, the result was comparable for small instances. For medium and big instances, superior results were reported with many feasible solutions found.

Liu et al. used a clique based algorithm for constructing feasible timetables

[121]. Their method was inspired by [45] where Carter and Johnson found many large cliques in the timetabling instances which may be useful for timetabling. The algorithm consists of three steps. In the first step, 45 cliques (corresponding to 45 time slots) were initialized. A clique is initialized by randomly choosing a vertex which is then expanded to include additional vertices before matching is run for room assignment. Vertices which were not included in a clique were handled in the next step. In the second step, the cliques were enlarged further before matching is run. In the third step, some vertices between two cliques were swapped. Better results were reported compared to other papers, especially for large instances.

Tables 3.13, 3.14 and 3.15 show the performance comparison among the state of the art methods on Hard:Small, Hard:Medium and Hard:Big instances respectively in terms of number of unassigned events. Ceshia et al. solved all of the Hard:Small instances to optimality in all runs [48]. Note that unlike Socha, ITC02 and ITC07 datasets, soft constraint violations are not considered for Hard instances. The authors are only concerned with finding feasible solutions for Hard instances which are relatively hard compared to the rest of the datasets. The details of the solvers are given in Table 3.16.

	Solver			
Instance	D1	D2	D3	D4
S1	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)
S2	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)
S3	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)
S4	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)
S5	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)
S6	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)
S7	**0**	**0**(**0.00**)	**0**(0.20)	**0**(**0.00**)
S8	**0**	**0**(1.90)	**0**(0.30)	**0**(**0.00**)
S9	**0**	**0**(3.85)	**0**(0.15)	**0**(**0.00**)
S10	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)
S11	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)
S12	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)
S13	**0**	**0**(1.00)	**0**(**0.00**)	**0**(**0.00**)
S14	**0**	3(5.95)	**0**(0.70)	**0**(**0.00**)
S15	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)
S16	**0**	**0**(**0.00**)	**0**(0.30)	**0**(**0.00**)
S17	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)
S18	**0**	**0**(0.45)	**0**(0.70)	**0**(**0.00**)
S19	**0**	**0**(1.20)	**0**(**0.00**)	**0**(**0.00**)
S20	**0**	**0**(**0.00**)	**0**(0.15)	**0**(**0.00**)

Table 3.13: Comparison among the state of the art methods on Hard small instances. Depicted is best(mean) of unassigned events. Note that some authors only reported their best results.

	Solver			
Instance	D1	D2	D3	D4
M1	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)
M2	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)
M3	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)
M4	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)
M5	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)
M6	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(0.90)
M7	14	1(4.15)	**0**(3.55)	**0**(**0.00**)
M8	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(0.30)
M9	2	**0**(4.90)	**0**(2.15)	**0**(0.35)
M10	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)
M11	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)
M12	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(0.60)
M13	**0**	**0** (0.50)	**0**(**0.00**)	**0**(**0.00**)
M14	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(0.05)
M15	**0**	**0**(0.05)	**0**(**0.00**)	**0**(**0.00**)
M16	1	1(5.15)	**0**(0.30)	**0**(**0.00**)
M17	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(0.15)
M18	**0**	**0**(6.05)	**0**(**0.00**)	**0**(0.30)
M19	**0**	**0**(5.45)	**0**(**0.00**)	**0**(0.50)
M20	3	2(10.60)	**0**(0.65)	**0**(0.55)

Table 3.14: Comparison among the state of the art methods on Hard medium instances. Depicted is best(mean) of unassigned events. Note that some authors only reported their best results.

	Solver			
Instance	D1	D2	D3	D4
B1	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(0.15)
B2	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(0.60)
B3	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(1.45)
B4	8	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)
B5	30	**0**(1.10)	1(3.20)	**0**(**0.00**)
B6	77	5(8.45)	10(15.40)	1(2.85)
B7	150	47(58.30)	39(46.65)	**21**(**29.25**)
B8	5	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)
B9	3	**0**(0.05)	**0**(**0.00**)	**0**(**0.00**)
B10	24	**0**(1.25)	**0**(1.95)	**0**(**0.00**)
B11	22	**0**(0.35)	**0**(2.35)	**0**(**0.00**)
B12	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(1.15)
B13	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(1.15)
B14	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(1.20)
B15	**0**	**0**(**0.00**)	**0**(**0.00**)	1(3.5)
B16	19	**0**(2.00)	**0**(**0.00**)	**0**(0.65)
B17	163	76(89.90)	**0**(**2.05**)	12(22.00)
B18	164	53(62.60)	**0**(1.70)	8(13.55)
B19	232	109(127.00)	40(53.20)	37(52.85)
B20	149	40(46.70)	9(14.05)	11(15.05)

Table 3.15: Comparison among the state of the art methods on Hard big instances. Depicted is best(mean) of unassigned events. Note that some authors only reported their best results.

Solver	Technique	Reference
D1	Genetic Algorithm	Lewis and Paechter
D2	Hybrid Simulated Annealing	Tuga et al.
D3	Clique Based Algorithm	Liu et al.
D4	Simulated Annealing	Ceshia et al.

Table 3.16: Details of solvers applied on Hard instances.

3.4 Conclusion

We have described the requirements of the course timetabling problem. The datasets (Socha, ITC02, ITC07, Hard) considered in this thesis are described such as the origin, source, time limit, hard and soft constraints. In addition, the instance specific statistics such as the # of events, # of rooms, # of features and # of students are provided. The formal presentation of the problem is

also presented. The related work for each dataset is reviewed and presented in a chronological order. State of the art methods are identified and studied. Generally, the best performing methods are based on SA. However, as evident in the literature, the method either requires extensive parameter tuning to obtain good results or is utilized on a limited number of instances/datasets.

Chapter 4

Finding Feasibility: Comparing GCH, MCTS and TS

In this chapter, we compare the effectiveness of several constructive heuristic approaches in finding feasible solutions for the course timetabling problem. We focus on MCTS as well as other state of the art methods.

4.1 Graph Coloring Heuristic (GCH)

GCH is a classical approach used in solving graph coloring problem. The heuristics derived are often utilized in timetabling problems. Often, the difficult events are assigned first so that the easier events will be successfully assigned later, when the environment becomes more constrained. We also include Dynamic Search Rearrangement (DSR) [112], a heuristic often used in constraint satisfaction problem in our experiments.

4.1.1 Algorithm Description

The heuristics are tested by using the procedure GCH as presented in Algorithm 19. It is a one pass method where, at each iteration, an event is selected and assigned to a selected place (or slot). The heuristics are described in Table 4.1. LD is considered as static ordering as the order of events is fixed throughout

the process meanwhile SD and DSR are dynamic as the next selected event is calculated at each iteration.

Algorithm 19

```
1: procedure GCH(best, unassignedE)
2:     remainingE ← unassignedE
3:     size ← |remainingE|
4:     for i = 1 to size do
5:         event ← selectEvent(remainingE)
6:         place ← selectPlace(event)
7:         if place is available then
8:             best ← best ∪ event                    ▷ assign event to place
9:         else
10:            unplacedE ← unplacedE ∪ event
11:        end if
12:        remainingE = remainingE − event
13:    end for
14:    unassignedE ← unplacedE
15: end procedure
```

Heuristics	Description
Random Ordering (RO)	*event* is selected randomly and assigned to a *place* (or *slot*) which is also selected randomly. It is a base for comparison with other heuristics.
Largest Degree (LD)	An *event* is selected randomly from the set E={events with the highest number of clashes with other events}. A *place* (or *slot*) is selected randomly from the set P={places (or slots) suitable for *event* and fit the least number of remaining events}.
Saturation Degree (SD)	An *event* is selected randomly from the set E_2={events with the highest number of clashes with other events} where $E_2 \subset E_1$ and E_1 ={events with the least number of suitable places (or slots)}. A *place* (or *slot*) is selected randomly from the set P={places (or slots) suitable for *event* and fit the least number of remaining events}.
Dynamic Search Rearrangement (DSR)	An *event* is selected randomly from the set E={events with the least number of suitable places (or slots)}. A *place* (or *slot*) is selected randomly from the set P={places (or slots) suitable for *event* and fit the least number of remaining events}.

Table 4.1: Heuristics

4.1.2 Experimental Results

We performed two types of assignment using the same set of heuristics. In the first type, events are assigned to places (specific slot and specific room). In the second type, events are assigned to slots with the aid of maximal matching for room assignment. For Socha instances, SD and LD are the most effective heuristics for the assignment by place and the assignment by slot respectively as shown in Table 4.2. Assignment by place using SD and assignment by slot using DSR are able to find feasible solution for all the instances. For ITC02, SD and LD work well for the assignment by place and the assignment by slot as presented in Table 4.3. It seems assignment by place is more suitable for Socha and ITC02 instances. The employment of heuristics is better for assignment by place compared to assignment by slot. In addition, the Socha and ITC02 instances are less constrained, therefore maximal matching for room assignment is trivial.

For ITC07, the DSR heuristic works well for both types of assignment as shown in Table 4.4. However, assignment by slot is more appropriate for these instances, despite the coarse employment of heuristics. Assignment by slot with DSR heuristic is able to find feasible solutions for 5 out of 24 instances. We believe the flexibility offered by maximal matching in room assignment is crucial for ITC07 instances which is more constrained compared to Socha and ITC02 instances.

	Assignment by Place				Assignment by Slot			
Instance	RO	LD	SD	DSR	RO	LD	SD	DSR
S1	0(0.00)	0(0.00)	0(0.00)	0(0.00)	0(0.00)	0(0.00)	0(0.00)	0(0.00)
S2	0(0.00)	0(0.00)	0(0.00)	0(0.00)	0(0.00)	0(0.00)	0(0.00)	0(0.00)
S3	0(0.00)	0(0.00)	0(0.00)	0(0.00)	0(0.00)	0(0.00)	0(0.00)	0(0.00)
S4	0(0.00)	0(0.00)	0(0.00)	0(0.00)	0(0.00)	0(0.00)	0(0.00)	0(0.00)
S5	0(0.00)	0(0.00)	0(0.00)	0(0.00)	0(0.00)	0(0.00)	0(0.00)	0(0.00)
M1	31(39.55)	0(0.00)	0(0.00)	0(0.00)	7(12.71)	0(0.00)	0(5.45)	0(5.00)
M2	32(40.13)	0(0.19)	0(0.00)	0(0.00)	9(15.45)	0(0.65)	0(2.97)	0(2.55)
M3	41(49.03)	0(0.03)	0(0.00)	0(0.00)	13(20.71)	0(0.00)	0(0.81)	0(0.48)
M4	28(34.16)	0(0.00)	0(0.00)	0(0.00)	8(11.71)	0(0.00)	0(2.39)	0(3.52)
M5	36(41.23)	0(0.32)	0(0.00)	0(0.00)	18(22.55)	0(0.00)	0(0.00)	0(0.68)
L	41(47.48)	1(4.87)	0(2.58)	1(6.94)	39(48.06)	2(6.65)	1(6.19)	0(6.48)
Avg.	(23.71)	(0.49)	(**0.23**)	(0.63)	(11.87)	(**0.66**)	(1.62)	(1.70)

Table 4.2: Comparison among assignment types with different heuristics on Socha instances. Depicted is best(mean) of unassigned events. n=31 runs.

	Assignment by Place				Assignment by Slot			
Instance	RO	LD	SD	DSR	RO	LD	SD	DSR
1	28(36.94)	0(0.00)	0(0.00)	0(0.00)	17(24.55)	0(0.00)	0(0.16)	0(0.23)
2	27(33.42)	0(0.00)	0(0.00)	0(0.00)	15(20.19)	0(0.00)	0(0.00)	0(0.00)
3	37(44.84)	0(0.00)	0(0.00)	0(0.00)	13(19.74)	0(0.00)	0(0.13)	0(0.39)
4	38(45.52)	0(0.06)	0(0.10)	0(0.03)	25(30.71)	0(0.03)	4(13.39)	0(7.26)
5	32(37.61)	0(0.06)	0(0.00)	0(0.10)	27(31.52)	0(0.03)	0(0.00)	0(0.03)
6	32(40.71)	0(0.00)	0(0.00)	0(0.00)	16(22.32)	0(0.00)	0(0.00)	0(0.74)
7	29(35.35)	0(0.00)	0(0.00)	0(0.00)	16(19.65)	0(0.00)	0(0.00)	0(0.00)
8	40(47.10)	0(0.00)	0(0.00)	0(0.00)	16(22.39)	0(0.00)	0(0.00)	0(0.29)
9	31(38.00)	0(0.00)	0(0.00)	0(0.00)	13(19.65)	0(0.00)	0(0.52)	0(0.42)
10	48(53.23)	0(0.19)	0(0.00)	0(0.00)	19(26.97)	0(0.90)	0(0.13)	0(1.16)
11	33(40.55)	0(0.06)	0(0.00)	0(0.00)	18(25.68)	0(0.00)	0(2.32)	0(4.68)
12	40(46.39)	0(0.00)	0(0.00)	0(0.00)	25(30.87)	0(0.00)	0(0.65)	0(0.19)
13	39(45.32)	0(0.00)	0(0.10)	0(0.03)	20(25.48)	0(0.19)	0(0.00)	0(0.77)
14	32(41.32)	0(0.00)	0(0.00)	0(0.00)	21(26.23)	0(0.00)	0(0.00)	0(0.00)
15	32(39.06)	0(0.00)	0(0.00)	0(0.00)	19(25.39)	0(0.00)	0(0.00)	0(0.00)
16	32(41.58)	0(0.00)	0(0.00)	0(0.00)	8(13.61)	0(0.00)	0(2.84)	0(2.48)
17	29(33.03)	0(0.03)	0(0.10)	0(0.35)	27(32.32)	0(0.00)	0(0.00)	0(0.65)
18	24(31.65)	0(0.00)	0(0.00)	0(0.00)	14(19.71)	0(0.00)	0(0.06)	0(0.03)
19	49(55.32)	0(0.03)	0(0.00)	0(0.00)	18(23.55)	0(3.10)	0(0.06)	0(0.94)
20	27(32.06)	0(0.00)	0(0.00)	0(0.00)	8(15.26)	0(0.00)	0(0.00)	0(0.00)
Avg.	(40.95)	(0.02)	(**0.01**)	(0.03)	(23.79)	(**0.21**)	(1.01)	(1.01)

Table 4.3: Comparison among assignment types with different heuristics on ITC02 instances. Depicted is best(mean) of unassigned events. n=31 runs.

	Assignment by Place				Assignment by Slot			
Instance	RO	LD	SD	DSR	RO	LD	SD	DSR
1	91(100.48)	40(57.87)	44(56.35)	47(57.58)	83(90.19)	45(52.90)	27(34.58)	26(34.29)
2	97(108.35)	53(66.52)	61(69.94)	53(64.55)	89(98.61)	53(63.90)	40(51.58)	36(45.03)
3	20(29.06)	6(10.23)	7(11.97)	7(12.26)	22(27.35)	2(7.42)	**0**(2.55)	**0**(4.35)
4	28(33.48)	11(15.00)	11(16.35)	12(16.61)	24(31.61)	9(13.74)	3(8.52)	4(9.94)
5	68(80.81)	28(37.13)	32(40.84)	32(41.68)	68(76.90)	27(33.35)	11(21.35)	13(20.42)
6	69(81.03)	28(38.29)	34(42.84)	33(43.81)	69(77.52)	20(30.65)	7(17.71)	11(20.26)
7	32(38.90)	19(26.00)	12(17.71)	15(19.23)	33(38.00)	17(22.26)	5(9.84)	6(11.87)
8	28(36.29)	12(19.26)	11(16.00)	9(15.45)	29(34.23)	10(17.65)	3(5.61)	**0**(4.97)
9	98(105.00)	55(69.68)	55(63.16)	55(61.65)	83(96.32)	58(67.58)	37(47.03)	32(43.26)
10	106(117.94)	76(86.32)	68(79.74)	72(79.81)	92(108.06)	73(84.94)	43(54.68)	43(53.29)
11	29(35.84)	7(12.74)	8(13.19)	9(13.84)	28(33.00)	7(12.26)	2(6.03)	1(7.10)
12	34(40.10)	14(18.68)	16(20.13)	15(19.68)	33(38.87)	11(18.26)	7(12.87)	4(14.26)
13	76(85.94)	27(41.23)	40(49.42)	35(48.19)	71(80.87)	25(33.71)	12(26.16)	17(25.26)
14	75(84.03)	35(44.32)	41(47.65)	39(46.16)	72(81.19)	40(45.97)	13(26.81)	16(24.81)
15	32(38.13)	11(17.71)	13(18.52)	12(18.39)	32(35.97)	13(19.23)	2(8.97)	5(9.97)
16	22(29.10)	5(7.45)	2(6.23)	4(6.55)	20(26.68)	5(8.03)	**0**(2.10)	**0**(2.39)
17	5(8.87)	**0**(1.45)	**0**(0.16)	**0**(0.35)	5(8.74)	**0**(1.13)	**0**(1.06)	**0**(1.26)
18	40(46.39)	14(20.23)	8(23.32)	14(24.03)	38(45.52)	8(20.13)	9(18.97)	10(17.61)
19	64(73.52)	35(45.35)	36(42.61)	30(41.81)	57(65.71)	33(40.19)	26(34.48)	23(30.23)
20	58(67.42)	2(9.29)	2(8.26)	3(8.90)	42(51.42)	2(5.10)	**0**(2.00)	**0**(2.52)
21	80(92.87)	26(36.58)	36(42.23)	32(39.65)	77(86.87)	23(36.19)	10(15.42)	10(14.65)
22	148(159.19)	106(117.23)	99(111.97)	98(110.23)	140(151.19)	97(107.48)	75(84.29)	63(74.16)
23	101(112.68)	49(59.61)	52(64.45)	51(63.39)	101(111.55)	48(63.97)	46(55.65)	40(52.81)
24	66(80.42)	30(40.03)	20(30.10)	21(28.13)	71(80.48)	30(38.00)	18(24.77)	13(21.94)
Avg.	(70.24)	(37.42)	(37.21)	(**36.75**)	(65.70)	(35.17)	(23.88)	(**22.78**)

Table 4.4: Comparison among assignment types with different heuristics on ITC07 instances. Depicted is best(mean) of unassigned events. n=31 runs.

4.2 Monte Carlo Tree Search (MCTS)

In MCTS, each node in the tree represents a state and each directed link represents an action leading to the state. Each node contains at least an average value and a visit count. MCTS consists of four main steps namely selection, expansion, simulation and back-propagation repeated many times within available resources. In the selection step, the tree is traversed from the root until a non-terminal node with unvisited action is reached. A new child node is added for the unvisited action in the expansion step. In the simulation step, a simulation is run from the child node to produce an outcome. The traversed nodes including the child node are updated with values from the outcome in the back-propagation step. Ultimately, the move made is the best child of the root node, which is usually the child node with the highest average value or highest visit count. The process is depicted in Figure 4.1, from [50].

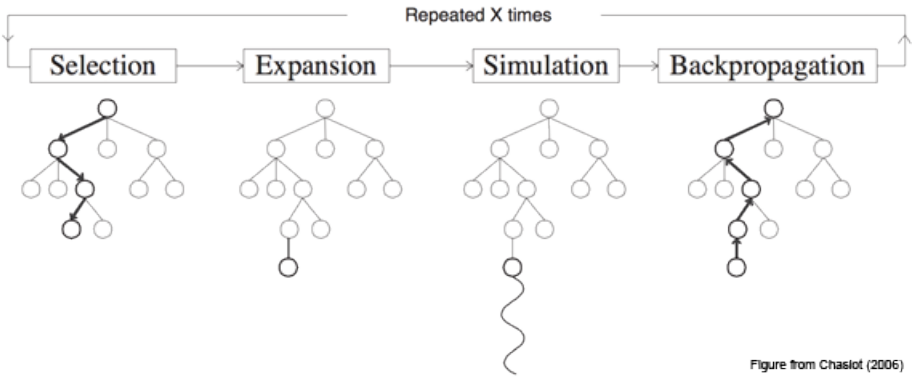

Figure 4.1: MCTS [50]

4.2.1 Algorithm Description

Assignment by slot (maximal matching for room assignment) is generally more effective compared to assignment by place as evident in GCH. Therefore, we employ assignment by slot in our MCTS implementation. The Node and Action class definition used are given in Figure 4.2.

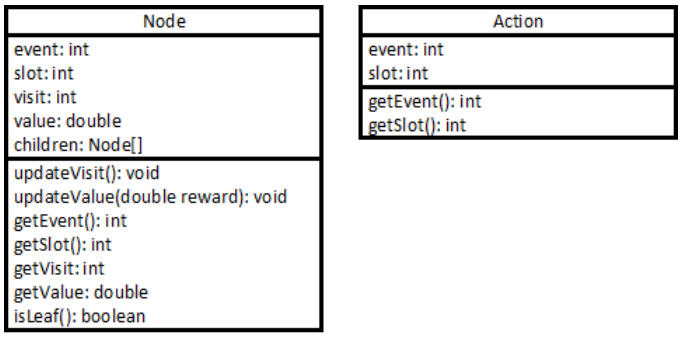

Figure 4.2: Node and Action class definition

The MCTS procedure is shown in Algorithm 20. The initial solution *initial*, is set to *best*. The list E is set to *unassignedE*. The best cost *f(best)* is initialized to the # of events in E. A node is created as the root. The exploration continues until a feasible solution is found (*unassignedE* is empty) or the elapsed time exceeds *runtime*. At the beginning of each exploration, *current* solution is set to *initial*, *remainingE* list is set to events in E and *visitedNode* list is set to empty. The root is added to *visitedNode*. *visitedNode* is used to keep nodes visited during tree traversal.

Algorithm 20

```
1:  procedure MCTS(best, unassignedE)
2:      initial ← best
3:      E ← unassignedE
4:      f(best) ← |E|
5:      create a node, root
6:      while unassignedE is not empty AND time.elapsed() < runtime do
7:          current ← initial
8:          remainingE ← E
9:          visitedNode ← empty
10:         visitedNode ← visitedNode ∪ root
11:
12:         treeGrowth(current, root, visitedNode, remainingE)
13:         reward ← simulation(current, best, f(best), unassignedE, remain-
    ingE, E)
14:         backpropagation(reward, visitedNode)
15:     end while
16: end procedure
```

In the treeGrowth procedure (Algorithm 21) , the *root* is set as the cur-

rent node, *currentNode*. While *currentNode* is not leaf node, one of the children of *currentNode* is selected by the SELECTION procedure (Algorithm 26) as the current node. It is then added to *visitedNode*. The event of *currentNode* is assigned to *current* according to the slot of *currentNode*. That event of *currentNode* is then removed from *remainingE*. If the *currentNode* is leaf node, we try to expand the tree from the leaf node. Possible actions are selected by the GETACTIONS procedure (Algorithm 27) and kept in the list of actions, A. If A is not empty, all actions in A are added as the children of *currentNode* by the EXPANSION PROCEDURE (Algorithm 30). One of the children will be chosen randomly as the child node, *childNode*. It is added to *visitedNode*. The event of *childNode* is assigned to *current* according to the slot of *childNode*. The event of *childNode* is then removed from *remainingE*.

Algorithm 21

```
1:  procedure treeGrowth(current, root, visitedNode, remainingE)
2:      currentNode ← root
3:      while currentNode is not leaf do
4:          currentNode ← selection(currentNode)
5:          visitedNode ← visitedNode ∪ currentNode
6:          current ← current ∪ currentNode.event          ▷ currentNode.slot
7:          remainingE ← remainingE − currentNode.event
8:      end while
9:      A ← getActions(remainingE, current)
10:     if A is not empty then
11:         expansion(currentNode, A)
12:         childNode ← select one of currentNode.children randomly
13:         visitedNode ← visitedNode ∪ childNode
14:         current ← current ∪ childNode.event            ▷ childNode.slot
15:         remainingE ← remainingE − childNode.event
16:     end if
17: end procedure
```

In the SIMULATION procedure (Algorithm 22), events are assigned to slots in *current* according to graph coloring heuristics. From experience in previous section, DSR is the most effective heuristic and therefore is used for event and slot selection. Events without any compatible slot are kept in *unplacedE*. *f(current)* is calculated as the # of events in *unplacedE*. If *f(current)* is better than *f(best)*, *best*, *f(best)* and *unassignedE* are updated. Reward is defined as the ratio of assigned events to # of events. Reward in the range of 0 to 1 is returned.

Algorithm 22

```
1: procedure SIMULATION(current, best, f(best), unassignedE, remainingE, E)
2:     size ← |remainingE|
3:     for i = 1 to size do
4:         event ← SELECTEVENT(remainingE, current)
5:         slot ← SELECTSLOT(event, remainingE, current)
6:         if slot available then
7:             current ← current ∪ event                ▷ assign event to slot
8:         else
9:             unplacedE ← unplacedE ∪ event
10:        end if
11:        remainingE ← remainingE − event
12:    end for
13:    f(current) ← |unplacedE|
14:    if f(current) < f(best) then
15:        best ← current
16:        f(best) ← f(current)
17:        unassignedE ← unplacedE
18:    end if
19:    return (|E| − |unplacedE|) / |E|
20: end procedure
```

Algorithm 23

```
1: procedure SELECTEVENT(remainingE, current)
2:     return an event in remainingE according to heuristics.
3: end procedure
```

Algorithm 24

```
1: procedure SELECTSLOT(event, remainingE, current)
2:     return a slot that is suitable for event.
3: end procedure
```

In the BACKPROPAGATION procedure (Algorithm 25), the visit and value property of each node in *visitedNode* are updated. The visit count is incremented while the value is updated as the cumulative mean of reward.

Algorithm 25

```
1: procedure BACKPROPAGATION(reward, visitedNode)
2:     for all node in visitedNode do
3:         node.updateVisit()
4:         node.updateValue(reward)
5:     end for
6: end procedure
```

The SELECTION, GETACTIONS and EXPANSION procedures mentioned earlier are described in details below. In the SELECTION procedure (Algorithm 26), a child among the children of *currentNode* with max UCB value is returned. B is a constant that is used to balance between exploration and exploitation of the search. When B is set higher, the less frequent visited nodes are given more priority. B is set as 0.0001.

Algorithm 26

1: **procedure** SELECTION(*currentNode*)
2: **return** $\arg\max_{i \in children\ of\ currentNode} value_i + B\sqrt{\frac{\ln visit_{currentNode}}{visit_i}}$
3: **end procedure**

In the GETACTIONS procedure (Algorithm 27), we employ several graph coloring heuristics to filter the possible actions to prevent the tree from getting too broad. In effect, the tree is pruned based on heuristics. Note that an action consists of an event and a slot.

Algorithm 27

```
1: procedure GETACTIONS(remainingE, current)
2:     A ← empty
3:     E ← GETEVENTS(remainingE, current)
4:     for all e in E do
5:         S ← GETSLOTS(e, remainingE, current)
6:         for all s in S do
7:             A ← A ∪ action(e, s)
8:         end for
9:     end for
10:    return A
11: end procedure
```

Algorithm 28

```
1: procedure getEvents(remainingE, current)
2:     return events in remainingE according to heuristics.
3: end procedure
```

Algorithm 29

```
1: procedure getSlots(e, remainingE, current)
2:     return slots that are suitable for e.
3: end procedure
```

In the expansion procedure (Algorithm 30), all actions in A are added as children of *currentNode*. Our implementation is different from the general MCTS where a child node is added whenever an unvisited action is encountered. Instead, multiple nodes are added. We intend to save computation cost at the expense of some memory.

Algorithm 30

```
1: procedure expansion(currentNode, A)
2:     add all actions in A as children of currentNode
3: end procedure
```

4.2.2 Experimental Results

4.2.2.1 Comparing Random Simulation and Heuristics Based Simulation

To make the simulation in MCTS more realistic, domain knowledge is incorporated into the play-outs [69, 163]. In this section, we compare the results of random simulation (random selection of events and slots) with heuristic based simulation. Dynamic Search Rearrangement (DSR) is used as it is the most effective heuristic as seen in the previous section.

100% feasibility is achieved for Socha and ITC02 instances when a heuristic is applied in the simulation phase of MCTS as shown in Table 4.5 and 4.6. MCTS with heuristic based simulation also performs better than MCTS with random simulation for all the ITC07 instances as shown in Table 4.7. Dash symbols in Table 4.6 and 4.7 indicate no result is available as the algorithm encountered insufficient heap memory in the tree growth phase of MCTS (even after extending

the default size from 256Mb to 1.5Gb). An error message was prompted indicating that the allocated 1.5Gb heap memory was exhausted during the runs. Note that the tree is expanded by using all the possible actions (each action involves assigning an event to a slot). On average, heuristic based simulation improves the results of random simulation for all the datasets considered.

	Simulation	
Instance	Random	DSR
S1	0(0.00)	0(0.00)
S2	0(0.00)	0(0.00)
S3	0(0.00)	0(0.00)
S4	0(0.00)	0(0.00)
S5	0(0.00)	0(0.00)
M1	3(4.42)	0(0.00)
M2	5(6.06)	0(0.00)
M3	8(10.71)	0(0.00)
M4	2(3.71)	0(0.00)
M5	10(13.10)	0(0.00)
L	34(35.94)	0(0.00)
Avg.	(6.72)	(**0.00**)

Table 4.5: Comparison between random and heuristics based simulation on Socha instances. Depicted is best(mean) of unassigned events. n=31 runs.

	Simulation	
Instance	Random	DSR
1	12(14.52)	0(0.00)
2	9(10.74)	0(0.00)
3	10(11.48)	0(0.00)
4	18(20.16)	0(0.00)
5	-	0(0.00)
6	-	0(0.00)
7	9(10.68)	0(0.00)
8	10(12.71)	0(0.00)
9	6(10.35)	0(0.00)
10	14(17.13)	0(0.00)
11	13(15.29)	0(0.00)
12	17(19.26)	0(0.00)
13	13(15.26)	0(0.00)
14	15(16.06)	0(0.00)
15	13(14.52)	0(0.00)
16	2(5.68)	0(0.00)
17	-	0(0.00)
18	7(10.00)	0(0.00)
19	12(13.90)	0(0.00)
20	5(6.35)	0(0.00)
Avg.	(13.18)	(**0.00**)

Table 4.6: Comparison between random and heuristics based simulation on ITC02 instances. Depicted is best(mean) of unassigned events. n=31 runs.

	Simulation	
Instance	Random	DSR
1	67(73.39)	14(15.90)
2	-	20(25.19)
3	-	0(0.00)
4	-	-
5	-	-
6	-	2(5.55)
7	-	-
8	-	0(0.00)
9	76(78.97)	22(24.77)
10	86(89.58)	29(32.58)
11	-	0(0.00)
12	-	-
13	-	7(9.39)
14	-	-
15	-	0(0.00)
16	-	0(0.00)
17	-	0(0.00)
18	-	-
19	-	-
20	34(36.58)	0(0.00)
21	65(67.23)	2(3.35)
22	127(131.39)	52(54.58)
23	-	-
24	-	-
Avg.	(79.52)	(**11.42**)

Table 4.7: Comparison between random and heuristics based simulation on ITC07 instances. Depicted is best(mean) of unassigned events. n=31 runs.

4.2.2.2 Tree Pruning Based on Heuristics

To solve the memory problem encountered earlier, we try to prune the tree in MCTS. Only certain actions (an action involves assigning an event to a slot) are considered in expanding the tree instead of all actions as in the previous section. In this section, we compare several heuristic based pruning mechanisms. The idea is drawn from the work in [100, 96, 14] where domain knowledge was utilized for pruning. The descriptions of the heuristics are presented in Table 4.8 based on Algorithm 27. Note that simulation based on DSR is used as it is more effective as shown previously.

Heuristics	Description
DSR	getEvents procedure returns a set of events with the least number of suitable slots, E=$\{e_1, e_2, ...e_m\}$. For each e_m in E, getSlots procedure returns slots suitable for e_m and fit the least number of remaining events, S=$\{s_1, s_2, ...s_n\}$. Each action(e_m, s_n) is added into a list for use in tree expansion.
LD-All	getEvents procedure returns a set of events with the highest number of clashes with other events, E=$\{e_1, e_2, ...e_m\}$. For each e_m in E, getSlots procedure returns all slots suitable for e_m, S=$\{s_1, s_2, ...s_n\}$. Each action(e_m, s_n) is added into a list for use in tree expansion.
MV-All	getEvents procedure returns a set of events with the least number of suitable slots, E=$\{e_1, e_2, ...e_m\}$. For each e_m in E, getSlots procedure returns all slots suitable for e_m, S=$\{s_1, s_2, ...s_n\}$. Each action(e_m, s_n) is added into a list for use in tree expansion.
SD-All	getEvents procedure returns a set of events with the highest number of clashes with other events, E_2=$\{e_1, e_2, ...e_m\}$ where $E_2 \subset E_1$ and E_1=$\{$events with the least number of suitable slots$\}$. For each e_m in E_2, getSlots procedure returns all slots suitable for e_m, S=$\{s_1, s_2, ...s_n\}$. Each action(e_m, s_n) is added into a list for use in tree expansion.

Table 4.8: Tree Pruning Heuristics

For Socha and ITC02 instances, 100% feasibility is achieved for all the heuristics tested for tree pruning as shown in Table 4.9 and 4.10. In fact the same feat is achieved even when no pruning is used as shown in the previous section indicating that these datasets are easy.

	Tree Pruning			
Instance	DSR	LD-All	MV-All	SD-All
S1	0(0.00)	0(0.00)	0(0.00)	0(0.00)
S2	0(0.00)	0(0.00)	0(0.00)	0(0.00)
S3	0(0.00)	0(0.00)	0(0.00)	0(0.00)
S4	0(0.00)	0(0.00)	0(0.00)	0(0.00)
S5	0(0.00)	0(0.00)	0(0.00)	0(0.00)
M1	0(0.00)	0(0.00)	0(0.00)	0(0.00)
M2	0(0.00)	0(0.00)	0(0.00)	0(0.00)
M3	0(0.00)	0(0.00)	0(0.00)	0(0.00)
M4	0(0.00)	0(0.00)	0(0.00)	0(0.00)
M5	0(0.00)	0(0.00)	0(0.00)	0(0.00)
L	0(0.00)	0(0.00)	0(0.00)	0(0.00)
Avg.	0(0.00)	0(0.00)	0(0.00)	0(0.00)

Table 4.9: Comparison among tree pruning heuristics on Socha instances. Depicted is best(mean) of unassigned events. n=31 runs.

	Tree Pruning			
Instance	DSR	LD-All	MV-All	SD-All
1	0(0.00)	0(0.00)	0(0.00)	0(0.00)
2	0(0.00)	0(0.00)	0(0.00)	0(0.00)
3	0(0.00)	0(0.00)	0(0.00)	0(0.00)
4	0(0.00)	0(0.00)	0(0.00)	0(0.00)
5	0(0.00)	0(0.00)	0(0.00)	0(0.00)
6	0(0.00)	0(0.00)	0(0.00)	0(0.00)
7	0(0.00)	0(0.00)	0(0.00)	0(0.00)
8	0(0.00)	0(0.00)	0(0.00)	0(0.00)
9	0(0.00)	0(0.00)	0(0.00)	0(0.00)
10	0(0.00)	0(0.00)	0(0.00)	0(0.00)
11	0(0.00)	0(0.00)	0(0.00)	0(0.00)
12	0(0.00)	0(0.00)	0(0.00)	0(0.00)
13	0(0.00)	0(0.00)	0(0.00)	0(0.00)
14	0(0.00)	0(0.00)	0(0.00)	0(0.00)
15	0(0.00)	0(0.00)	0(0.00)	0(0.00)
16	0(0.00)	0(0.00)	0(0.00)	0(0.00)
17	0(0.00)	0(0.00)	0(0.00)	0(0.00)
18	0(0.00)	0(0.00)	0(0.00)	0(0.00)
19	0(0.00)	0(0.00)	0(0.00)	0(0.00)
20	0(0.00)	0(0.00)	0(0.00)	0(0.00)
Avg.	0(0.00)	0(0.00)	0(0.00)	0(0.00)

Table 4.10: Comparison among tree pruning heuristics on ITC02 instances. Depicted is best(mean) of unassigned events. n=31 runs.

For ITC07 instances, memory problems are no longer encountered when tree pruning is used. Table 4.11 shows the maximum heap memory committed during 31 runs for each problematic instance in the previous section. Obviously, the maximum heap memory committed is far below the allocated size of 1.5Gb. Note that the heap memory sizes are measured using Java Monitoring and Management Console, a utility supplied by Java Development Kit (JDK).

Instance	Maximum Heap Memory Committed (Mb)
4	27.24
5	130.57
7	24.08
12	56.40
14	147.10
18	27.82
19	158.65
23	178.08
24	134.30

Table 4.11: The maximum heap memory (Mb) committed during 31 runs for selected ITC07 instances.

MV-All is the most effective pruning heuristic where feasible solutions are found for all instances except instances 1, 2, 9, 10 and 22 as shown in Table 4.12. Interestingly, this is consistent with the result of a Constraint Programming approach [41] which was unable to find feasible solutions for instances 1, 2, 9 and 10. Instance 22 was not considered by the author. Generally, better results are observed when pruning is used regardless of heuristics used. Tree size is greatly reduced with pruning. As a result, pruning allows the search to focus more time on better choices by eliminating obviously poor choices.

	Tree Pruning			
Instance	DSR	LD-All	MV-All	SD-All
1	3(6.94)	3(7.13)	3(6.39)	3(6.87)
2	7(12.77)	11(16.77)	6(11.19)	9(12.13)
3	0(0.00)	0(0.00)	0(0.00)	0(0.00)
4	0(0.00)	0(0.00)	0(0.00)	0(0.00)
5	0(1.84)	0(0.23)	0(0.03)	0(0.16)
6	0(0.90)	0(0.61)	0(0.29)	0(0.42)
7	0(0.13)	0(0.00)	0(0.00)	0(0.03)
8	0(0.00)	0(0.00)	0(0.00)	0(0.00)
9	10(14.39)	13(17.61)	14(16.97)	9(15.16)
10	14(18.74)	17(24.48)	15(19.71)	16(21.77)
11	0(0.00)	0(0.00)	0(0.00)	0(0.00)
12	0(1.35)	0(0.00)	0(0.00)	0(0.06)
13	0(2.58)	1(2.29)	0(1.13)	0(1.26)
14	1(3.29)	0(1.55)	0(0.84)	0(2.19)
15	0(0.10)	0(0.06)	0(0.00)	0(0.00)
16	0(0.00)	0(0.00)	0(0.00)	0(0.00)
17	0(0.00)	0(0.00)	0(0.00)	0(0.00)
18	0(0.68)	0(0.00)	0(0.00)	0(0.00)
19	6(10.61)	1(3.16)	0(2.77)	4(8.26)
20	0(0.00)	0(0.00)	0(0.00)	0(0.00)
21	0(0.74)	0(1.03)	0(1.10)	0(0.39)
22	28(34.87)	46(49.06)	41(44.68)	31(40.45)
23	2(8.77)	7(13.97)	0(6.16)	2(9.23)
24	1(4.42)	0(0.35)	0(0.06)	0(1.61)
Avg.	(5.13)	(5.76)	(**4.64**)	(5.00)

Table 4.12: Comparison among tree pruning heuristics on ITC07 instances. Depicted is best(mean) of unassigned events. n=31 runs.

We tested several values for B as given in Table 4.13. Note that tree pruning based on heuristics (MV-All) and simulation based on DSR are used here. The value of B determines the exploration and exploitation of the MCTS. A higher value will allow the search to explore more and exploit less relatively. For the search to work effectively, different values of B (exploration and exploitation) are required for different instances.

	Values of B	
Instance	0.00001	0.0001
1	3(6.52)	3(6.39)
2	7(11.32)	6(11.19)
3	0(0.00)	0(0.00)
4	0(0.00)	0(0.00)
5	0(0.03)	0(0.03)
6	0(0.23)	0(0.29)
7	0(0.00)	0(0.00)
8	0(0.00)	0(0.00)
9	8(16.26)	14(16.97)
10	16(19.52)	15(19.71)
11	0(0.00)	0(0.00)
12	0(0.00)	0(0.00)
13	0(0.81)	0(1.13)
14	0(0.87)	0(0.84)
15	0(0.00)	0(0.00)
16	0(0.00)	0(0.00)
17	0(0.00)	0(0.00)
18	0(0.00)	0(0.00)
19	0(3.03)	0(2.77)
20	0(0.00)	0(0.00)
21	0(0.97)	0(1.10)
22	39(44.74)	41(44.68)
23	1(5.55)	0(6.16)
24	0(0.26)	0(0.06)
Avg.	(**4.59**)	(4.64)

Table 4.13: The results of setting different values for B on ITC07 instances. Depicted is best(mean) of unassigned events. n=31 runs.

4.2.2.3 Extended Runtime for MCTS

We extended the runtime for MCTS on selected ITC07 instances which we could not find a feasible solution previously. As shown in Table 4.14, the results improve when the runtime is extended for both values of B. In addition, the appropriate value of B for MCTS to work effectively depends on the allocated *runtime*. The value of 0.00001 seems to be more effective for the shorter runtime whereas 0.0001 is more suitable for the longer runtime. With extended runtime of 5T and B=0.0001, MCTS was able to find feasible solutions for instances 1, 2 and 9. However, it still could not a find feasible solution for instances 10 and 22.

	Values of B			
	0.00001		0.0001	
	runtime		*runtime*	
Instance	T	5T	T	5T
1	3(6.52)	0(**1.10**)	3(6.39)	0(**1.10**)
2	7(11.32)	0(3.52)	6(11.19)	0(**2.97**)
9	8(16.26)	2(7.13)	14(16.97)	0(**5.87**)
10	16(19.52)	6(10.13)	15(19.71)	5(**9.81**)
22	39(44.74)	21(29.29)	41(44.68)	19(**28.23**)
Avg.	(19.67)	(10.23)	(19.79)	(**9.59**)

Table 4.14: Comparison between MCTS with runtime of T and $5\,T$ for selected ITC07 instances. Depicted is best(mean) of unassigned events. n=31 runs.

4.3 Tabu Search (TS)

TS selects the best admissible neighborhood move (non-tabu or allowed by aspiration). The move with the lowest cost function is selected even if it increases the cost function of the current solution. If the current solution is better than the best solution, the best solution is updated. The reversal of the selected move is then set tabu for some time to prevent cycling.

4.3.1 Algorithm Description

PARTIALCOL [22], which was initially used for solving graph coloring problems, was adapted by [53], [169] and [118] in solving course timetabling problems. The TS procedure, presented in Algorithm 31, is based on PARTIALCOL. A neighbor move involves moving an event from the list of unplaced events *unplacedE* to a time slot in the current solution *current*. At the start of each iteration, all the neighborhood moves are evaluated (line 7-21) by considering all non-tabu suitable time slots for all the events in *unplacedE*.

Event e is temporarily removed from *unplacedE*. To feasibly move e into a particular time slot, events conflicting with e (violated clash or precedence constraint) are temporarily moved from *current* to *unplacedE*. By comparison, [118] only removed events which violated a clash constraint from the time slots. As maximal matching is computationally expensive, it is used for room assignment only when necessary. If matching could not find a room for the event under

consideration, a room is chosen randomly from among the suitable rooms and the related event is moved from *current* to *unplacedE*.

Algorithm 31

```
1: procedure TS(best, unassignedE)
2:     unplacedE ← unassignedE
3:     current ← best
4:     f(best) ← f(current)
5:     while unplacedE is not empty AND time.elapsed() < runtime  do
6:         min ← ∞
7:         for all e ∈ unplacedE do
8:             unplacedE ← unplacedE − e
9:             for all s ∈ S | S non-tabu slot suitable for e do
10:                current ← current − {events conflicting e}
11:                unplacedE ← unplacedE ∪ {events conflicting e}
12:                if f(candidate) < min then
13:                    bestEvent ← e
14:                    bestSlot ← s
15:                    min ← f(candidate)
16:                end if
17:                unplacedE ← unplacedE − {events conflicting e}
18:                current ← current ∪ {events conflicting e}
19:            end for
20:            unplacedE ← unplacedE ∪ e
21:        end for
22:        current ← current − {events conflicting bestEvent}
23:        current ← current ∪ bestEvent                          ▷ bestSlot
24:        f(current) ← min
25:        if f(current) < f(best) then
26:            best ← current
27:            f(best) ← f(current)
28:            unassignedE ← unplacedE
29:        end if
30:        set tabu {events conflicting bestEvent} from original time slots
31:        unplacedE ← unplacedE − bestEvent
32:        unplacedE ← unplacedE ∪ {events conflicting bestEvent}
33:    end while
34: end procedure
```

The cost function f used to evaluate solutions (*current*, *candidate*, *best*) is based on the number of unplaced events (Equation 4.1):

$$\sum_{e \in unplacedE} 1 \tag{4.1}$$

The candidate solution with the lowest number of unplaced events is preferred. As a comparison, the cost function used in [169] was the number of students required to attend the unplaced events (Equation 4.2):

$$\sum_{e \in unplacedE} size[e] \tag{4.2}$$

The events conflicting with e are moved back from *unplacedE* to *current* before evaluating the next non-tabu suitable time slot for the event under consideration. When all the non-tabu time slots are evaluated, e is placed back to *unplacedE* before the next event is considered. Ultimately, the neighbor move with the lowest candidate cost *f(candidate)* is recorded as *bestEvent* and *bestSlot*.

Events conflicting with *bestEvent* are extracted from *current* (line 22). The best neighbor move is applied where the *bestEvent* is moved to the *bestSlot* of *current* (line 23). *best*, *f(best)* and *unassignedE* are updated if *f(current)* is better than *f(best)*. The events conflicting with *bestEvent* are set tabu from returning to their original time slots for a number of iterations (line 30) according to the tabu tenure (Equation 4.3):

$$\text{RANDOM}[10) + |unplacedE| \tag{4.3}$$

where $|unplacedE|$ is the number of unplaced events. A value of 10 is used in the random element of the tabu tenure length. We use this value as the same value was used in [22], [118] and [169]. The value works well for all the datasets that we consider. The value of tabu tenure determines the level of exploration for the search. When the value of tabu tenure is set too high, most of the available moves are not reachable and may restrict the search. When the value is too low, cycling tends to occur which may stall the search.

bestEvent is removed from *unplacedE* while all the events conflicting with *bestEvent* are added to *unplacedE*. The iteration continues until *unplacedE* is empty (feasible solution is found) or the elapsed time exceeds *runtime*.

4.4 Comparing GCH, MCTS and TS

In this section, we compare the most competitive variant of GCH, MCTS and TS in terms of average results. The GCH variant compared here is based on

assignment by slot and the DSR heuristic. Meanwhile, the MCTS variant compared utilizes MV-All heuristics for tree pruning and DSR heuristic for the simulation phase. Feasible solutions are found for all Socha instances by all three algorithms as shown in Table 4.15. 100% feasibility is achieved by both MCTS and TS.

All three algorithms are able to find feasible solutions for all ITC02 instances as shown in Table 4.16. However, only MCTS manages to obtain 100% feasibility for all the instances. TS achieves 100% feasibility for all the instances except instance 7 (87%).

Results comparison for ITC07 instances is given in Table 4.17. TS is the only algorithm capable of finding feasible solutions for all the instances. GCH found feasible solutions for 5 out of the 24 instances (instances 3, 8, 16, 17 and 20). Meanwhile, MCTS found feasible solutions for all the instances except instances 1, 2, 9, 10 and 22. In fact, none of the algorithms are able to achieve 100% feasibility for all the instances. TS performed well with 100% feasibility for all the instances except instances 11, 19 and 23.

	GCH (Slot DSR)			MCTS			TS		
		Unassigned			Unassigned			Unassigned	
Instance	Fea.(%)	best	mean	Fea.(%)	best	mean	Fea.(%)	best	mean
S1	100	0	0.00	100	0	0.00	100	0	0.00
S2	100	0	0.00	100	0	0.00	100	0	0.00
S3	100	0	0.00	100	0	0.00	100	0	0.00
S4	100	0	0.00	100	0	0.00	100	0	0.00
S5	100	0	0.00	100	0	0.00	100	0	0.00
M1	6	0	5.00	100	0	0.00	100	0	0.00
M2	29	0	2.55	100	0	0.00	100	0	0.00
M3	74	0	0.48	100	0	0.00	100	0	0.00
M4	10	0	3.52	100	0	0.00	100	0	0.00
M5	71	0	0.68	100	0	0.00	100	0	0.00
L	3	0	6.48	100	0	0.00	100	0	0.00

Table 4.15: Comparison among GCH, MCTS and TS on Socha instances. n=31 runs.

	GCH (Slot DSR)			MCTS			TS		
		Unassigned			Unassigned			Unassigned	
Instance	Fea.(%)	best	mean	Fea.(%)	best	mean	Fea.(%)	best	mean
1	84	0	0.23	100	0	0.00	100	0	0.00
2	100	0	0.00	100	0	0.00	100	0	0.00
3	81	0	0.39	100	0	0.00	100	0	0.00
4	10	0	7.26	100	0	0.00	100	0	0.00
5	97	0	0.03	100	0	0.00	100	0	0.00
6	74	0	0.74	100	0	0.00	100	0	0.00
7	100	0	0.00	100	0	0.00	87	0	0.13
8	90	0	0.29	100	0	0.00	100	0	0.00
9	81	0	0.42	100	0	0.00	100	0	0.00
10	42	0	1.16	100	0	0.00	100	0	0.00
11	10	0	4.68	100	0	0.00	100	0	0.00
12	87	0	0.19	100	0	0.00	100	0	0.00
13	60	0	0.77	100	0	0.00	100	0	0.00
14	100	0	0.00	100	0	0.00	100	0	0.00
15	100	0	0.00	100	0	0.00	100	0	0.00
16	32	0	2.48	100	0	0.00	100	0	0.00
17	71	0	0.65	100	0	0.00	100	0	0.00
18	97	0	0.03	100	0	0.00	100	0	0.00
19	65	0	0.94	100	0	0.00	100	0	0.00
20	100	0	0.00	100	0	0.00	100	0	0.00

Table 4.16: Comparison among GCH, MCTS and TS on ITC02 instances. n=31 runs.

	GCH (Slot DSR)			MCTS			TS		
		Unassigned			Unassigned			Unassigned	
Instance	Fea.(%)	best	mean	Fea.(%)	best	mean	Fea.(%)	best	mean
1	0	26	34.29	0	3	6.39	100	0	0.00
2	0	36	45.03	0	6	11.19	100	0	0.00
3	10	0	4.35	100	0	0.00	100	0	0.00
4	0	4	9.94	100	0	0.00	100	0	0.00
5	0	13	20.42	97	0	0.03	100	0	0.00
6	0	11	20.26	72	0	0.29	100	0	0.00
7	0	6	11.87	100	0	0.00	100	0	0.00
8	3	0	4.97	100	0	0.00	100	0	0.00
9	0	32	43.26	0	14	16.97	100	0	0.00
10	0	43	53.29	0	15	19.71	100	0	0.00
11	0	1	7.10	100	0	0.00	87	0	0.26
12	0	4	14.26	100	0	0.00	100	0	0.00
13	0	17	25.26	23	0	1.13	100	0	0.00
14	0	16	24.81	42	0	0.84	100	0	0.00
15	0	5	9.97	100	0	0.00	100	0	0.00
16	10	0	2.39	100	0	0.00	100	0	0.00
17	42	0	1.26	100	0	0.00	100	0	0.00
18	0	10	17.61	100	0	0.00	100	0	0.00
19	0	23	30.23	6	0	2.77	81	0	0.29
20	13	0	2.52	100	0	0.00	100	0	0.00
21	0	10	14.65	13	0	1.10	100	0	0.00
22	0	63	74.16	0	41	44.68	100	0	0.00
23	0	40	52.81	3	0	6.16	94	0	0.06
24	0	13	21.94	94	0	0.06	100	0	0.00

Table 4.17: Comparison among GCH, MCTS and TS on ITC07 instances. n=31 runs.

4.5 Discussion

Assignment by slot is more effective than assignment by place as evident in the GCH implementation. Therefore, our MCTS implementation is designed based on assignment by slot. Events are assigned to empty rooms. Maximal matching is used for room assignment only when necessary.

We faced heap memory problem when the tree is expanded by all the possible actions at one time. In fact, we decided to do so instead of expanding the tree by one action at a time to save computational cost. This is essential as the timetabling problem that we are solving is constrained by a time limit, due to competition rules. We addressed the heap memory problem by pruning the tree, based on heuristics. Tree pruning limits the number of nodes added to the tree (hence heap memory utilization) but it may also cut off the paths to good solutions.

MCTS works best for games like Go but not for timetabling problem. MCTS lacks the flexibility offered by local search algorithms (e.g. TS). In each exploration of MCTS, events are assigned sequentially in a constructive manner where moves made cannot be undone or redone. This fits well for games such as Go but not for timetabling where events are allowed to be unassigned and reassigned. In effect, the connectivity of search space provided by MCTS is inferior to that of a local search for timetabling problem.

The rigid tree structure of MCTS also limits the effort to hybridize the algorithm with local search. In fact, local search played a vital role in obtaining good results for similar learning based algorithm (ACO).

The time limit imposed on timetabling problems also restrict the use of learning based algorithms (MCTS) which usually require reasonable computational resources to perform effectively.

4.6 Conclusion

We compared several graph coloring heuristics (LD, SD, DSR) using two types of assignments, namely assignment by place and assignment by slot. We found that the variant based on slot assignment and DSR heuristic is the most effective in terms of average means of unassigned events for all the datasets considered.

We compared random and heuristic based simulation (DSR) for the simulation part of MCTS. It seems heuristic based simulation is better than random simulation. Heuristics make simulation more realistic compared to random simulation where events and slots are selected randomly.

We tested several tree pruning heuristics (DSR, LD-All, MV-All and SD-All). Tree pruning improves the effectiveness of MTCS in finding feasible solutions in terms of average number of unassigned events. Empirical results also show that MV-All worked best among the heuristics tested.

We compared the best variant of GCH, MCTS and TS in finding feasible solutions. MCTS worked well for Socha and ITC02 instances but was lacking in terms of performance for ITC07 instances. MCTS could not find a feasible solution for instances 10 and 22 of ITC07 even with extended runtime. Computational experience shows that the value of B (selection part of MCTS) can affect the results. For MCTS to work effectively, this value needs to be adjusted for specific instances. In addition, the suitable value also depends on the *runtime*. Overall, TS shows great potential and therefore we decided to focus on improving the algorithm, which is our focus in the next chapter.

Chapter 5

Finding Feasibility: TSSP-ILS Algorithm

In a comparison to GCH and MCTS, TS is the best method in finding feasible solutions. Therefore, we focus on improving TS. In this chapter, we propose several enhancements to the algorithm. The work in this chapter has been published as [88]:

- Say Leng Goh, Graham Kendall, Nasser R. Sabar. Improved Local Search Approaches to Solve Post Enrolment Course Timetabling Problem. European Journal of Operational Research, 2017.

and submitted for peer review:

- Say Leng Goh, Graham Kendall, Nasser R. Sabar. A Hybrid Local Search Approach for Hard Course Timetabling Problem. Information Systems and Operational Research (INFOR). Under review.

5.1 Tabu Search with Sampling and Perturbation (TSSP)

TS overcomes getting stuck in local optima by selecting the best non-tabu move, even if it increases the cost function of the current solution. However, TS is susceptible to cycling which may cause the search to get stuck. Therefore, our enhancements below are devised with the general aim of alleviating cycling.

5.1.1 Algorithm Description

The procedure is shown in Algorithm 32. It is important to note that no parameter tuning is required in this algorithm.

Algorithm 32

```
1: procedure TSSP(best, unassignedE)
2:     unplacedE ← unassignedE
3:     current ← best
4:     f(best) ← f(current)
5:     ITER ← room^3
6:     i ← 0
7:     while unplacedE is not empty AND time.elapsed() < runtime do
8:         sampleE ← select S events randomly from unplacedE
9:         min ← ∞
10:        for all e ∈ sampleE do
11:            unplacedE ← unplacedE − e
12:            for all s ∈ S | S non-tabu slot suitable for e do
13:                current ← current − {events conflicting e}
14:                unplacedE ← unplacedE ∪ {events conflicting e}
15:                if f(candidate) < min then
16:                    bestEvent ← e
17:                    bestSlot ← s
18:                    min ← f(candidate)
19:                end if
20:                unplacedE ← unplacedE − {events conflicting e}
21:                current ← current ∪ {events conflicting e}
22:            end for
23:            unplacedE ← unplacedE ∪ e
24:        end for
25:        current ← current − {events conflicting bestEvent}
26:        current ← current ∪ bestEvent                          ▷ bestSlot
27:        f(current) ← min
28:        if f(current) < f(best) then
29:            best ← current
30:            f(best) ← f(current)
31:            unassignedE ← unplacedE
32:        end if
33:        set tabu {events conflicting bestEvent} from original time slots
34:        unplacedE ← unplacedE − bestEvent
35:        unplacedE ← unplacedE ∪ {events conflicting bestEvent}
36:        if i = ITER then
37:            PERTURB(current)
38:            i ← 0
39:            reset tabu list
40:        end if
41:        i = i + 1
42:    end while
43: end procedure
```

Instead of evaluating all the non-tabu time slots for all the events, we only evaluate the non-tabu time slots for certain sampled events. At the start of the iteration, S events are selected randomly from *unplacedE* and added to *sampleE* list (line 8). The event sampling size S is set as [0.25% $\times$ # of events]. It is proportional to the # of events, e.g. event sampling size is 1, 2 and 3 for the # of events between 1-400, 401-800 and 801-1200 respectively.

Rather than using the cost function based solely on the number of unplaced events to evaluate solutions (*current*, *candidate* and *best*), we propose a novel cost function f which is based on the number of unplaced events plus the clash ratio (Eq. 5.1):

$$\sum_{e \in unplacedE} 1 + \frac{clash[e]}{clashSum} \tag{5.1}$$

where $clash[e]$ is the clash number of e with other events and *clashSum* is the total clash number of all events. Effectively, the candidate solution with the lowest number of unplaced events is preferred and ties broken using the clash number.

Unlike other implementations which tracked idle iterations before performing perturbation, we perturbed the current solution at certain iteration intervals $ITER$ (lines 36-40). If $i = ITER$, *current* is perturbed, i is reset to 0 and tabu list is reset. In the PERTURB procedure (Algorithm 33), we attempt to move each assigned event to each time slot (except the time slot currently occupied by the event) in *slotList* (shuffled randomly) by using either a Swap or a Kempe operator. Maximal matching is used sparingly for room assignment. The event is moved only if the time slot under consideration is suitable for the event (not violating any hard constraints). Perturbation is used to explore the search space and does not affect the current solution as no event is extracted. However, when used too often it may slow down the search of a feasible solution. $ITER$ is set as $room^3$ (line 5). Essentially, the search is allowed to progress longer when the search space is larger before perturbation is initiated.

Algorithm 33

```
1: procedure PERTURB(solution)
2:     for all e ∈ solution do
3:         SHUFFLE(slotList)
4:         for all slot ∈ slotList do
5:             if RANDOM[0, 2) = 1 then
6:                 if SWAP(solution, e, slot) then
7:                     break;
8:                 end if
9:             else
10:                if KEMPE(solution, e, slot) then
11:                    break;
12:                end if
13:            end if
14:        end for
15:    end for
16: end procedure
```

The neighborhood structures used in the PERTURB procedure are:

- Swap: A swap is attempted between e with event in each room (room list shuffled randomly) in *slot* . A swap is carried out if all the hard constraints are satisfied.

- Kempe: Kempe chain interchange is attempted [170], [52], [118]. A chain is built between events in the time slot occupied by e (time slot A) and events in *slot* (time slot B). Initially, e is added to the chain. Next, events in time slot B clashing with e are added to the chain. Next, events in time slot A clashing with the chained events in time slot B are added to the chain. Then, events in time slot B clashing with the chained events in time slot A are added to the chain. The process is repeated until a complete chain is obtained. Subsequently, the chained events in time slot A are moved to time slot B and vice versa, assuming that all the hard constraints are satisfied.

As in TS, the algorithm is exited when a feasible solution is found or the runtime is exceeded. Therefore, the time or iteration number to find a feasible solution varies for each instance. The time limit is determined by running a program[1] on the executing machine (our machine is entitled to 190s).

[1] http://www.idsia.ch/Files/ttcomp2002/ Last accessed: June 13, 2017.

5.1.2 Experimental Results

5.1.2.1 The Effect of Event Sampling

We compared a few event sampling size S proportional to # of events, in TS on selected instances. 100% feasibility is achieved for all the proportions on all the instances tested. However, S=[0.25% × # of events] recorded the lowest time to feasibility for all the instances as shown in Table 5.1.

	Event Sampling Size S		
Instance	0.25% × # of Events	0.5% × # of Events	1% × # of Events
Socha-L	0.1868	0.3494	0.4887
ITC02-1	0.0223	0.0481	0.0677
ITC07-1	0.4858	0.8161	1.4368
Hard-S1	0.0061	0.0065	0.0100
Hard-M1	0.0232	0.0355	0.0655
Hard-B1	0.1613	0.2674	0.4971
Avg.	0.1476	0.2538	0.4276

Table 5.1: The results of setting different event sampling size S upon time to feasibility (s) on selected instances. n=31 runs.

In this section, we compare TS with and without event sampling. Note that the cost function used here is given in Eq. 4.1. Event sampling size S=[0.25% × # of events] is compared with no sampling at one continuum end and event sampling size S=1 at another continuum end. As evident in Table 5.2, TS with event sampling, regardless of type, is more effective than TS without sampling for all the datasets in terms of the average number of unassigned events.

			Event Sampling Size S	
Dataset	# of Events	No Sampling	1	0.25% × # of Events
Socha	100-400	0.00	0.00	0.00
ITC02	350-440	0.01	0.00	0.00
ITC07	100-600	0.03	0.00	0.00
Hard small	200-225	1.87	1.24	1.27
Hard medium	390-425	1.69	0.35	0.39
Hard big	1000-1075	17.14	11.03	8.56

Table 5.2: Comparison of average number of unassigned events among event sampling on datasets.

For event sampling size S=[0.25% × # of events], S is equivalent to 1, 2 and 3 for # of events between 1-400, 401-800 and 801-1200 respectively. As the #

of events in Socha and Hard small instances are less than or equal to 400, S are equivalent for both types of event sampling, which is 1. Therefore, the average results for S=1 and S=[0.25% × # of events], are comparable for these instances.

Meanwhile, the # of events for ITC02, ITC07, Hard medium and Hard big instances varies between 100-1075. Clearly, event sampling size S=[0.25% × # of events], is more effective than $S = 1$ when the # of events is high (Hard big). The average time to feasibility is shown in Table 5.3. Dash symbols in the table indicate that no valid average time is available as there are unassigned events.

As TS with event sampling size S=[0.25% × # of events] performs well in terms of average number of unassigned events and average time to feasibility, it is used from here on.

			Event Sampling Size S	
Dataset	# of Events	No Sampling	1	0.25% × # of Events
Socha	100-400	1.0923	0.0243	0.0326
ITC02	350-440	-	-	0.0408
ITC07	100-600	-	0.8040	0.8007
Hard small	200-225	-	-	-
Hard medium	390-425	-	-	-
Hard big	1000-1075	-	-	-

Table 5.3: Comparison of average time to feasibility (s) between TS with and without event sampling on datasets.

5.1.2.2 The Effect of Cost Functions and Perturbation

We tested several values for *ITER* as given in Table 5.4. The value of *room* recorded the lowest means for all the selected instances. Therefore, it is used onwards.

	ITER		
Instance	$room$	$room^2$	$room^3$
Socha-L	0.1345	0.0406	0.0281
ITC02-1	0.1432	0.0381	0.0216
ITC07-1	0.8371	0.4171	0.1671
Hard-S1	0.0245	0.0106	0.0065
Hard-M1	0.0929	0.0265	0.0171
Hard-B1	0.6923	0.1716	0.1481
Avg.	0.3208	0.1174	0.0647

Table 5.4: The results of setting different values for *ITER* upon time to feasibility (s) on selected instances. n=31 runs.

Next, we compare the effect of using different cost functions with or without perturbation on TS with event sampling.

Without perturbation, the cost function in Eq. 5.1 is effective in terms of the average number of unassigned events compared to the other cost functions for all the datasets, as shown in Table 5.5. Likewise, the cost function is effective in terms of average time to feasibility when used without perturbation (Table 5.6). Dash symbols indicate that the average time to feasibility is invalid because there are unassigned events.

When perturbation is paired with the cost function in Eq. 5.1, TS with event sampling performs the best where the average number of unassigned events is the lowest for all the datasets as presented in Table 5.5. In fact, 100% feasibility is achieved for all the datasets except Hard big. The combination also produces the lowest average time to feasibility for all the datasets as shown in Table 5.6.

Perturbation did not improve the results for the cost functions in Eq. 4.1 (Hard medium and Hard big instances) and 4.2 (Hard small and Hard big instances) as shown in Table 5.5. The extent of perturbation employed is redundant for these cost functions as both already have good diversification capability. Consequently, the exploitation capability of the search is weakened.

	Cost Functions					
	Eq. 4.1		Eq. 4.2		Eq. 5.1	
Dataset	-	Perturbation	-	Perturbation	-	Perturbation
Socha	0.00	0.00	0.00	0.00	0.00	0.00
ITC02	0.00	0.00	0.07	0.00	0.00	0.00
ITC07	0.00	0.00	0.00	0.00	0.00	0.00
Hard small	1.27	0.73	0.47	0.69	0.20	0.00
Hard medium	0.39	0.44	0.26	0.07	0.10	0.00
Hard big	8.56	10.82	7.86	8.51	4.19	3.60

Table 5.5: Comparison of average number of unassigned events among cost functions with and without perturbation on datasets.

	Cost Functions					
	Eq. 4.1		Eq. 4.2		Eq. 5.1	
Dataset	-	Perturbation	-	Perturbation	-	Perturbation
Socha	0.0326	0.0205	0.0139	0.0146	0.0132	0.0133
ITC02	0.0408	0.0197	-	0.0271	0.0211	0.0190
ITC07	0.8007	0.5612	0.2610	0.3113	0.2021	0.1953
Hard small	-	-	-	-	-	0.4834
Hard medium	-	-	-	-	-	0.8211
Hard big	-	-	-	-	-	-

Table 5.6: Comparison of average time to feasibility (s) among cost functions with and without perturbation on datasets.

5.1.2.3 Comparing TS and TSSP

We compare the performance of TS and TSSP in finding feasible solutions in terms of the number of unassigned events and average time to feasibility.

For Socha instances, both algorithms performed well with 100% feasibility. However, TSSP is faster as shown in Table 5.7. On average, the algorithm managed to find feasible solutions in less than one-tenth of a second. The p values (less than 0.05) reveal a significant difference between the means (time to feasibility) of TS and TSSP for all the instances. It is important to note that as we performed multiple statistical tests, some tests are significant by chance or false positives, at the $p<0.05$ level. For example, Socha dataset with 11 instances may have about 0.55 (0.05×11) instance with false positives. Meanwhile, ITC07 dataset with 24 instances may have about 1.20 (0.05×24) false positives. Datasets with 20 instances such as ITC02, Hard big, Hard medium and Hard small may have about 1.00 (0.05×20) false positive each. This condition applies to all the t-tests performed in this thesis.

Instance	TS	TSSP	t-test (p value)
S1	0.0361	0.0032	0.000
S2	0.0268	0.0019	0.000
S3	0.0290	0.0013	0.000
S4	0.0397	0.0019	0.000
S5	0.0313	0.0019	0.000
M1	2.2906	0.0210	0.000
M2	2.0184	0.0242	0.000
M3	1.9681	0.0194	0.000
M4	1.8355	0.0203	0.000
M5	2.0655	0.0219	0.000
L	1.6742	0.0287	0.000

Table 5.7: Comparison of average time to feasibility (s) between TS and TSSP on Socha instances. n=31 runs.

For ITC02, TSSP is more effective than TS in terms of feasibility and the number of unassigned events as shown in Table 5.8. TSSP achieved 100% feasibility for all the instances. As a comparison, TS achieved 100% feasibility for all the instances except instance 7 (87%). The p value of 0.039 (less than 0.05) reveals a significant difference between the means (unassigned events) of TS and TSSP for instance 7. Note that the rest of the instances are omitted in Table 5.8 as they have means equivalent to 0. In addition, TSSP is faster than

TS as shown in Table 5.9. On average, the algorithm managed to find feasible solutions in less than one-tenth of a second. The p values (less than 0.05) reveal a significant difference between the means (time to feasibility) of TS and TSSP for all the instances. Note that dash symbols in Table 5.9 indicate that the average time to feasibility is invalid (feasibility is not 100%) and therefore p value is invalid.

	TS			TSSP			
		Unassigned			Unassigned		t-test
Instance	Fea.(%)	best	mean	Fea.(%)	best	mean	(p value)
7	87	0	0.13	100	0	0.00	0.039

Table 5.8: Comparison between TS and TSSP on ITC02 instances. n=31 runs.

Instance	TS	TSSP	t-test (p value)
1	1.7419	0.0239	0.000
2	1.7355	0.0200	0.000
3	1.1168	0.0142	0.000
4	1.7348	0.0203	0.000
5	0.6163	0.0103	0.000
6	0.9790	0.0148	0.000
7	-	0.0181	-
8	1.8723	0.0213	0.000
9	1.9865	0.0319	0.000
10	1.5881	0.0232	0.000
11	1.8161	0.0203	0.000
12	1.5023	0.0226	0.000
13	1.5506	0.0177	0.000
14	1.0461	0.0148	0.000
15	1.0119	0.0135	0.000
16	2.1765	0.0339	0.000
17	0.4819	0.0084	0.000
18	1.5652	0.0187	0.000
19	1.4961	0.0200	0.000
20	1.0377	0.0129	0.000

Table 5.9: Comparison of average time to feasibility (s) between TS and TSSP on ITC02 instances. n=31 runs.

For ITC07, TSSP performed better than TS as shown in Table 5.10. TSSP managed to achieve 100% feasibility for all the instances. Meanwhile, TS achieved 100% feasibility for all the instances except instance 11 (87%), instance 19 (81%) and instance 23 (94%). The p value of 0.015 (less than 0.05)

reveals a significant difference between the means (unassigned events) of TS and TSSP for instance 19. TSSP is also faster than TS as shown in Table 5.11. The algorithm managed to obtain feasible solutions in less than one second except instance 22. The p values (less than 0.05) reveal a significant difference between the means (time to feasibility) of TS and TSSP for all the instances except instance 3.

	TS			TSSP			
		Unassigned			Unassigned		t-test
Instance	Fea.(%)	best	mean	Fea.(%)	best	mean	(p value)
11	87	0	0.26	100	0	0.00	0.053
19	81	0	0.29	100	0	0.00	0.015
23	94	0	0.06	100	0	0.00	0.156

Table 5.10: Comparison between TS and TSSP on ITC07 instances. n=31 runs.

Instance	TS	TSSP	t-test (p value)
1	3.0639	0.1797	0.000
2	10.0226	0.4126	0.000
3	3.1600	0.0055	0.299
4	0.1277	0.0142	0.000
5	1.1787	0.0229	0.000
6	1.1800	0.0281	0.000
7	0.3955	0.0090	0.000
8	0.1310	0.0052	0.000
9	30.1587	0.4516	0.000
10	27.8168	0.8026	0.000
11	-	0.0119	-
12	1.2755	0.0161	0.045
13	1.3423	0.0355	0.000
14	1.2823	0.0313	0.000
15	0.0906	0.0058	0.000
16	0.0855	0.0032	0.000
17	0.0232	0.0013	0.000
18	0.1894	0.0129	0.000
19	-	0.2139	-
20	0.9274	0.0181	0.000
21	1.9823	0.0690	0.000
22	61.9365	2.1113	0.000
23	-	0.1894	-
24	2.0416	0.0371	0.000

Table 5.11: Comparison of average time to feasibility (s) between TS and TSSP on ITC07 instances. n=31 runs.

For Hard small instances, TSSP performed relatively better than TS as shown in Table 5.12. TSSP managed to achieve 100% feasibility for all the instances. In contrast, TS fails to find feasible solution for instances S7, S12, S14, S18 and S20. The p values (less than 0.05) reveal a significant difference between the means (unassigned events) of TS and TSSP for all the instances except for instances S10 and S19. TSSP is also quicker than TS as shown in Table 5.13. The p values (less than 0.05) reveal a significant difference between the means (time to feasibility) of TS and TSSP for all the instances.

	TS			TSSP			
		Unassigned			Unassigned		t-test
Instance	Fea.(%)	best	mean	Fea.(%)	best	mean	(p value)
S3	16	0	1.39	100	0	0.00	0.000
S5	3	0	2.81	100	0	0.00	0.000
S7	0	1	3.13	100	0	0.00	0.000
S8	6	0	2.74	100	0	0.00	0.000
S9	6	0	1.74	100	0	0.00	0.000
S10	90	0	0.10	100	0	0.00	0.078
S12	0	2	2.03	100	0	0.00	0.000
S13	61	0	0.55	100	0	0.00	0.001
S14	0	4	6.45	100	0	0.00	0.000
S17	35	0	0.84	100	0	0.00	0.000
S18	0	5	7.87	100	0	0.00	0.000
S19	97	0	0.03	100	0	0.00	0.321
S20	0	7	7.71	100	0	0.00	0.000

Table 5.12: Comparison between TS and TSSP on Hard small instances. n=31 runs.

Instance	TS	TSSP	t-test (p value)
S1	0.1700	0.0081	0.000
S2	0.2371	0.0058	0.000
S3	-	0.0135	-
S4	0.1929	0.0094	0.000
S5	-	0.0406	-
S6	0.2442	0.0071	0.000
S7	-	0.0187	-
S8	-	2.5910	-
S9	-	2.1845	-
S10	-	0.0165	-
S11	0.2290	0.0065	0.000
S12	-	0.0110	-
S13	-	0.2539	-
S14	-	4.3123	-
S15	0.1474	0.0065	0.000
S16	0.1226	0.0077	0.000
S17	-	0.0213	-
S18	-	0.0910	-
S19	-	0.0284	-
S20	-	0.0335	-

Table 5.13: Comparison of average time to feasibility (s) between TS and TSSP on Hard small instances. n=31 runs.

For Hard medium instances, TSSP is shown to be better in terms of performance than TS in Table 5.14. TSSP managed to achieve 100% feasibility for all the instances. Meanwhile, TS could not find feasible solution for instances M7, M9, M16 and M20. The p values (less than 0.05) reveal a significant difference between the means (unassigned events) of TS and TSSP for all the instances except instance M12. TSSP is also quicker than TS as shown in Table 5.15. The p values (less than 0.05) reveal a significant difference between the means (time to feasibility) of TS and TSSP for all the instances except M11.

	TS			TSSP			
		Unassigned			Unassigned		t-test
Instance	Fea.(%)	best	mean	Fea.(%)	best	mean	(p value)
M5	90	0	0.13	100	0	0.00	0.098
M7	0	1	3.71	100	0	0.00	0.000
M9	0	1	4.77	100	0	0.00	0.000
M12	97	0	0.06	100	0	0.00	0.321
M13	16	0	2.58	100	0	0.00	0.000
M14	26	0	1.94	100	0	0.00	0.000
M15	48	0	1.45	100	0	0.00	0.000
M16	0	1	4.61	100	0	0.00	0.000
M18	16	0	2.52	100	0	0.00	0.000
M19	10	0	4.90	100	0	0.00	0.000
M20	0	3	7.16	100	0	0.00	0.000

Table 5.14: Comparison between TS and TSSP on Hard medium instances. n=31 runs.

Instance	TS	TSSP	t-test (p value)
M1	1.0255	0.0229	0.000
M2	1.2877	0.0203	0.000
M3	1.6606	0.0306	0.000
M4	1.7703	0.1316	0.000
M5	-	0.2177	-
M6	4.4906	0.1903	0.000
M7	-	7.1881	-
M8	4.1361	0.0906	0.000
M9	-	6.7658	-
M10	1.3348	0.0168	0.000
M11	7.4665	0.0648	0.144
M12	-	0.0290	-
M13	-	0.2577	-
M14	-	0.0513	-
M15	-	0.3839	-
M16	-	0.3652	-
M17	4.5945	0.0681	0.024
M18	-	0.1448	-
M19	-	0.1900	-
M20	-	0.1923	-

Table 5.15: Comparison of average time to feasibility (s) between TS and TSSP on Hard medium instances. n=31 runs.

For Hard big instances, TSSP is outstanding compared to TS as presented in

Table 5.16. 100% feasibility is achieved by TSSP for 14 out of the 20 instances. TSSP is able to find feasible solution for 15 out of 20 instances. Meanwhile, TS fails to find feasible solution for 6 out of 20 instances. The p values (less than 0.05) reveal a significant difference between the means (unassigned events) of TS and TSSP for all the instances except for instances B9 and B15. TSSP is also faster than TS as shown in Table 5.17. The p values (less than 0.05) reveal a significant difference between the means (time to feasibility) of TS and TSSP for all the instances.

	TS			TSSP			
		Unassigned			Unassigned		t-test
Instance	Fea.(%)	best	mean	Fea.(%)	best	mean	(p value)
B5	6	0	2.35	100	0	0.00	0.000
B6	0	15	19.16	0	2	4.19	0.000
B7	0	48	60.06	0	28	36.65	0.000
B9	97	0	0.03	100	0	0.00	0.321
B10	3	0	2.84	100	0	0.00	0.000
B11	6	0	2.61	100	0	0.00	0.000
B15	81	0	0.81	100	0	0.00	0.068
B16	74	0	0.74	100	0	0.00	0.010
B17	0	37	47.32	0	1	6.23	0.000
B18	0	37	47.32	23	0	1.42	0.000
B19	0	94	113.65	0	7	19.94	0.000
B20	0	38	45.97	0	2	7.16	0.000

Table 5.16: Comparison between TS and TSSP on Hard big instances. n=31 runs.

Instance	TS	TSSP	t-test (p value)
B1	16.5013	0.1561	0.000
B2	16.9668	0.2865	0.000
B3	19.5135	0.2581	0.000
B4	66.0226	3.8416	0.000
B5	-	26.4042	-
B6	-	-	-
B7	-	-	-
B8	64.6487	3.6406	0.000
B9	-	2.3458	-
B10	-	27.2355	-
B11	-	39.0497	-
B12	17.8629	0.2984	0.000
B13	29.0042	0.6887	0.000
B14	20.0426	0.3884	0.000
B15	-	4.5716	-
B16	-	4.9458	-
B17	-	-	-
B18	-	-	-
B19	-	-	-
B20	-	-	-

Table 5.17: Comparison of average time to feasibility (s) between TS and TSSP on Hard big instances. n=31 runs.

As TSSP is shown to be more effective, our focus will be on TSSP from here on.

5.1.2.4 Comparing TSSP with State of the Art Methods

Our method performed generally faster on average time (especially instances 10, 19, 23 and 24) than the Improved PARTIALCOL by Lewis [118] while being equally effective (100% feasibility) in finding feasible solutions as shown in Table 5.18. As a comparison, the allowed time for Lewis's method was 247s.

	LS-Colouring [41]		I. PARTIALCOL [118]		TSSP	
Instance	Time(s)	Fea.(%)	Time(s)	Fea.(%)	Time(s)	Fea.(%)
1	7.31	100	0.25	100	0.18	100
2	15.80	100	0.79	100	0.41	100
3	0.47	100	0.02	100	0.01	100
4	0.48	100	0.02	100	0.01	100
5	2.77	100	0.06	100	0.02	100
6	3.47	100	0.08	100	0.03	100
7	0.59	100	0.03	100	0.01	100
8	0.49	100	0.01	100	0.01	100
9	14.78	100	0.68	100	0.45	100
10	53.87	98	2.03	100	0.80	100
11	0.63	100	0.03	100	0.01	100
12	0.73	100	0.04	100	0.02	100
13	3.86	100	0.08	100	0.04	100
14	3.75	100	0.11	100	0.03	100
15	0.60	100	0.01	100	0.01	100
16	0.50	100	0.01	100	0.00	100
17	-	-	0.00	100	0.00	100
18	-	-	0.02	100	0.01	100
19	-	-	0.71	100	0.21	100
20	-	-	0.01	100	0.02	100
21	-	-	0.08	100	0.07	100
22	-	-	3.80	100	2.11	100
23	-	-	1.10	100	0.19	100
24	-	-	0.18	100	0.04	100

Table 5.18: Comparing TSSP with other solvers on ITC07. n=31 runs.

5.2 Hybrid of TSSP and ILS

In this section, we propose to hybridize TSSP with Iterated Local Search (ILS) to find feasible solutions. As opposed to the previous setting where we allow TSSP to run until *runtime* is exceeded, we now allocate only $\frac{3}{4}$ of *runtime* for TSSP and the rest for ILS.

5.2.1 Algorithm Description

TSSP-ILS is shown in Algorithm 34.

Algorithm 34

```
1: procedure TSSP-ILS(best, unassignedE)
2:     TSSP(best, unassignedE)
3:     if unassignedE is not empty then
4:         ILS(best, unassignedE)
5:     end if
6: end procedure
```

If $\frac{3}{4}$ of *runtime* is exceeded and no feasible solution is found by TSSP (there are remaining events in *unassignedE*), the best solution recorded so far is passed to ILS for further processing until *runtime* is exceeded. The algorithm is exited early whenever a feasible solution is found (*unassignedE* is empty).

ILS is presented in Algorithm 35. At the start of each iteration, *best* is perturbed (line 3) where assigned events are shuffled randomly by using the PERTURB procedure given in Algorithm 33. Next, swap is attempted between each unassigned event *e1* in *unassignedE* with each assigned event *e2* in *best* (line 4-13). Swap will be carried out if the clash number of *e2* is less than the clash number of *e1* and the time slot occupied by *e2* is suitable for *e1* (not violating any hard constraints). As a result, *unassignedE* will have a pool of easier events to be assigned later. *unassignedE* is sorted by clash number in descending order (line 14). In effect, harder events will be scheduled first. Transfer is attempted for each event in *unassignedE* to each time slot in *best* (line 15-22). The event e will be moved to the time slot if no hard constraint is violated. The iteration stops when *unassignedE* is empty (feasible solution is found) or *runtime* is reached.

Algorithm 35

```
1: procedure ILS(best, unassignedE)
2:     while unassignedE is not empty AND time.elapsed() < runtime do
3:         PERTURB(best)
4:         for all e1 ∈ unassignedE do
5:             for all e2 ∈ best do
6:                 if clash[e2] < clash[e1] then
7:                     if slot of e2 is suitable for e1 then
8:                         swap e1 and e2
9:                         break
10:                    end if
11:                end if
12:            end for
13:        end for
14:        sort unassignedE by the clash number (descending order)
15:        for all e ∈ unassignedE do
16:            for slot = 1 to 45 do
17:                if slot is suitable for e then
18:                    move e from unassignedE to slot of best
19:                    break
20:                end if
21:            end for
22:        end for
23:    end while
24: end procedure
```

5.2.2 Experimental Results

5.2.2.1 Comparing TSSP and TSSP-ILS

In this section, we compare the effectiveness of TSSP and TSSP-ILS in finding feasible solutions. Both methods performed equally well on all the Hard small and Hard medium instances with 100% feasibility. However, TSSP-ILS is more effective than TSSP on the Hard big instances in terms of feasibility and the number of unassigned events as shown in Table 5.19. TSSP-ILS is able to find feasible solutions for 17 instances compared to 15 for TSSP. It improves the feasibility % for the instances B6, B17 and B18. In addition, TSSP-ILS improves the mean of unassigned events for the instances B6, B7, B17, B18, B19 and B20. We conducted a t-test to compare the means between both algorithms for each instance and the respective p values are given in the last column of the table. The dash symbols indicates that t cannot be computed for the particular instance because the standard deviations of both groups are 0 (means are 0).

The p values (less than 0.05) indicated a significant difference between the means (unassigned events) of TSSP and TSSP-ILS for all the instances except B18, B19 and B20. We did not conduct t-test on the Hard small and Hard medium instances as all means are 0 for both TSSP and TSSP-ILS.

	TSSP			TSSP-ILS			
		Unassigned			Unassigned		t-test
Instance	Fea.(%)	best	mean	Fea.(%)	best	mean	p value
B1	100	0	0.00	100	0	0.00	-
B2	100	0	0.00	100	0	0.00	-
B3	100	0	0.00	100	0	0.00	-
B4	100	0	0.00	100	0	0.00	-
B5	100	0	0.00	100	0	0.00	-
B6	0	2	4.19	16	0	1.87	0.000
B7	0	28	36.65	0	23	29.55	0.000
B8	100	0	0.00	100	0	0.00	-
B9	100	0	0.00	100	0	0.00	-
B10	100	0	0.00	100	0	0.00	-
B11	100	0	0.00	100	0	0.00	-
B12	100	0	0.00	100	0	0.00	-
B13	100	0	0.00	100	0	0.00	-
B14	100	0	0.00	100	0	0.00	-
B15	100	0	0.00	100	0	0.00	-
B16	100	0	0.00	100	0	0.00	-
B17	0	1	6.23	6	0	3.97	0.000
B18	23	0	1.42	48	0	0.84	0.058
B19	0	7	19.94	0	1	16.97	0.155
B20	0	2	7.16	0	2	6.65	0.477

Table 5.19: Comparison between TSSP and TSSP-ILS on Hard big instances. n=31 runs.

5.2.2.2 Comparing ILS and TSSP-ILS

We also compare the effectiveness of ILS and TSSP-ILS in finding feasible solutions. Obviously, TSSP-ILS is more effective than ILS on all the small, medium and big instances in terms of feasibility and the number of unassigned events as shown in Table 5.20, 5.21 and 5.22. The p values of t-test revealed a significant difference between the means (unassigned events) of ILS and TSSP-ILS for all the instances except S8.

	ILS			TSSP-ILS			
		Unassigned			Unassigned		t-test
Instance	Fea.(%)	best	mean	Fea.(%)	best	mean	(p value)
S1	100	0	0.00	100	0	0.00	-
S2	100	0	0.00	100	0	0.00	-
S3	100	0	0.00	100	0	0.00	-
S4	100	0	0.00	100	0	0.00	-
S5	100	0	0.00	100	0	0.00	-
S6	100	0	0.00	100	0	0.00	-
S7	100	0	0.00	100	0	0.00	-
S8	97	0	0.03	100	0	0.00	0.321
S9	10	0	4.65	100	0	0.00	0.000
S10	100	0	0.00	100	0	0.00	-
S11	100	0	0.00	100	0	0.00	-
S12	100	0	0.00	100	0	0.00	-
S13	45	0	2.52	100	0	0.00	0.000
S14	0	12	20.16	100	0	0.00	0.000
S15	100	0	0.00	100	0	0.00	-
S16	100	0	0.00	100	0	0.00	-
S17	100	0	0.00	100	0	0.00	-
S18	100	0	0.00	100	0	0.00	-
S19	3	0	27.77	100	0	0.00	0.000
S20	100	0	0.00	100	0	0.00	-

Table 5.20: Comparison between ILS and TSSP-ILS on Hard small instances. n=31 runs.

	ILS			TSSP-ILS			
		Unassigned			Unassigned		t-test
Instance	Fea.(%)	best	mean	Fea.(%)	best	mean	(p value)
M1	100	0	0.00	100	0	0.00	-
M2	100	0	0.00	100	0	0.00	-
M3	100	0	0.00	100	0	0.00	-
M4	100	0	0.00	100	0	0.00	-
M5	100	0	0.00	100	0	0.00	-
M6	19	0	11.87	100	0	0.00	0.000
M7	0	72	79.26	100	0	0.00	0.000
M8	0	21	31.71	100	0	0.00	0.000
M9	0	29	40.19	100	0	0.00	0.000
M10	100	0	0.00	100	0	0.00	-
M11	100	0	0.00	100	0	0.00	-
M12	100	0	0.00	100	0	0.00	-
M13	100	0	0.00	100	0	0.00	-
M14	100	0	0.00	100	0	0.00	-
M15	100	0	0.00	100	0	0.00	-
M16	0	57	75.81	100	0	0.00	0.000
M17	100	0	0.00	100	0	0.00	-
M18	0	24	61.32	100	0	0.00	0.000
M19	10	0	42.13	100	0	0.00	0.000
M20	48	0	8.45	100	0	0.00	0.000

Table 5.21: Comparison between ILS and TSSP-ILS on Hard medium instances. n=31 runs.

	ILS			TSSP-ILS			
		Unassigned			Unassigned		t-test
Instance	Fea.(%)	best	mean	Fea.(%)	best	mean	(p value)
B1	100	0	0.00	100	0	0.00	-
B2	100	0	0.00	100	0	0.00	-
B3	100	0	0.00	100	0	0.00	-
B4	100	0	0.00	100	0	0.00	-
B5	100	0	0.00	100	0	0.00	-
B6	0	70	90.13	16	0	1.87	0.000
B7	0	185	191.65	0	23	29.55	0.000
B8	100	0	0.00	100	0	0.00	-
B9	100	0	0.00	100	0	0.00	-
B10	100	0	0.00	100	0	0.00	-
B11	100	0	0.00	100	0	0.00	-
B12	100	0	0.00	100	0	0.00	-
B13	100	0	0.00	100	0	0.00	-
B14	100	0	0.00	100	0	0.00	-
B15	0	130	143.48	100	0	0.00	0.000
B16	100	0	0.00	100	0	0.00	-
B17	0	313	325.48	6	0	3.97	0.000
B18	0	181	195.65	48	0	0.84	0.000
B19	0	301	309.32	0	1	16.97	0.000
B20	0	167	188.23	0	2	6.65	0.000

Table 5.22: Comparison between ILS and TSSP-ILS on Hard big instances. n=31 runs.

5.2.2.3 Comparing TSSP-ILS with State of the Art Methods

In this section, we compare the performance of TSSP-ILS with the state of the art methods in the literature. As evident from Table 5.23 and Table 5.24, TSSP-ILS is superior compared to other methods for Hard small and Hard medium instances. TSSP-ILS also outperformed other methods for Hard big instances as shown in Table 5.25. We managed to find feasible solutions for 17 out of the 20 instances using $runtime = T$ (190s). Details of the solvers are given in Table 5.26.

	Solvers				
Instance	D1	D2	D3	D4	TSSP-ILS
S1	**0**	**0**(**0.00**)	**0**(**0.00**)	-	**0**(**0.00**)
S2	**0**	**0**(**0.00**)	**0**(**0.00**)	-	**0**(**0.00**)
S3	**0**	**0**(**0.00**)	**0**(**0.00**)	-	**0**(**0.00**)
S4	**0**	**0**(**0.00**)	**0**(**0.00**)	-	**0**(**0.00**)
S5	**0**	**0**(**0.00**)	**0**(**0.00**)	-	**0**(**0.00**)
S6	**0**	**0**(**0.00**)	**0**(**0.00**)	-	**0**(**0.00**)
S7	**0**	**0**(**0.00**)	**0**(0.20)	-	**0**(**0.00**)
S8	**0**	**0**(1.90)	**0**(0.30)	-	**0**(**0.00**)
S9	**0**	**0**(3.85)	**0**(0.15)	-	**0**(**0.00**)
S10	**0**	**0**(**0.00**)	**0**(**0.00**)	-	**0**(**0.00**)
S11	**0**	**0**(**0.00**)	**0**(**0.00**)	-	**0**(**0.00**)
S12	**0**	**0**(**0.00**)	**0**(**0.00**)	-	**0**(**0.00**)
S13	**0**	**0**(1.00)	**0**(**0.00**)	-	**0**(**0.00**)
S14	**0**	3(5.95)	**0**(0.70)	-	**0**(**0.00**)
S15	**0**	**0**(**0.00**)	**0**(**0.00**)	-	**0**(**0.00**)
S16	**0**	**0**(**0.00**)	**0**(0.30)	-	**0**(**0.00**)
S17	**0**	**0**(**0.00**)	**0**(**0.00**)	-	**0**(**0.00**)
S18	**0**	**0**(0.45)	**0**(0.70)	-	**0**(**0.00**)
S19	**0**	**0**(1.20)	**0**(**0.00**)	-	**0**(**0.00**)
S20	**0**	**0**(**0.00**)	**0**(0.15)	-	**0**(**0.00**)

Table 5.23: Comparing TSSP-ILS with other solvers on Hard small instances. Depicted is best(mean) of unassigned events. n=31 runs.

	Solvers				
Instance	D1	D2	D3	D4	TSSP-ILS
M1	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)
M2	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)
M3	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)
M4	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)
M5	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)
M6	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(0.90)	**0**(**0.00**)
M7	14	1(4.15)	**0**(3.55)	**0**(**0.00**)	**0**(**0.00**)
M8	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(0.30)	**0**(**0.00**)
M9	2	**0**(4.90)	**0**(2.15)	**0**(0.35)	**0**(**0.00**)
M10	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)
M11	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)
M12	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(0.60)	**0**(**0.00**)
M13	**0**	**0** (0.50)	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)
M14	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(0.05)	**0**(**0.00**)
M15	**0**	**0**(0.05)	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)
M16	1	1(5.15)	**0**(0.30)	**0**(**0.00**)	**0**(**0.00**)
M17	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(0.15)	**0**(**0.00**)
M18	**0**	**0**(6.05)	**0**(**0.00**)	**0**(0.30)	**0**(**0.00**)
M19	**0**	**0**(5.45)	**0**(**0.00**)	**0**(0.50)	**0**(**0.00**)
M20	3	2(10.60)	**0**(0.65)	**0**(0.55)	**0**(**0.00**)

Table 5.24: Comparing TSSP-ILS with other solvers on Hard medium instances. Depicted is best(mean) of unassigned events. n=31 runs.

	Solvers				
Instance	D1	D2	D3	D4	TSSP-ILS
B1	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(0.15)	**0**(**0.00**)
B2	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(0.60)	**0**(**0.00**)
B3	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(1.45)	**0**(**0.00**)
B4	8	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)
B5	30	**0**(1.10)	1(3.20)	**0**(**0.00**)	**0**(**0.00**)
B6	77	5(8.45)	10(15.40)	1(2.85)	**0**(**1.87**)
B7	150	47(58.30)	39(46.65)	**21**(**29.25**)	23(29.55)
B8	5	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)
B9	3	**0**(0.05)	**0**(**0.00**)	**0**(**0.00**)	**0**(**0.00**)
B10	24	**0**(1.25)	**0**(1.95)	**0**(**0.00**)	**0**(**0.00**)
B11	22	**0**(0.35)	**0**(2.35)	**0**(**0.00**)	**0**(**0.00**)
B12	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(1.15)	**0**(**0.00**)
B13	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(1.15)	**0**(**0.00**)
B14	**0**	**0**(**0.00**)	**0**(**0.00**)	**0**(1.20)	**0**(**0.00**)
B15	**0**	**0**(**0.00**)	**0**(**0.00**)	1(3.5)	**0**(**0.00**)
B16	19	**0**(2.00)	**0**(**0.00**)	**0**(0.65)	**0**(**0.00**)
B17	163	76(89.90)	**0**(**2.05**)	12(22.00)	**0**(3.97)
B18	164	53(62.60)	**0**(1.70)	8(13.55)	**0**(**0.84**)
B19	232	109(127.00)	40(53.20)	37(52.85)	**1**(**16.97**)
B20	149	40(46.70)	9(14.05)	11(15.05)	**2**(**6.65**)

Table 5.25: Comparing TSSP-ILS with other solvers on Hard big instances. Depicted is best(mean) of unassigned events. n=31 runs.

Solver	Technique	Reference
D1	GA	Lewis and Paechter
D2	Hybrid SA	Tuga et al.
D3	Clique Based Algorithm	Liu et al.
D4	SA	Ceshia et al.

Table 5.26: Details of solvers applied on Hard instances.

5.2.2.4 Extended Runtime for TSSP-ILS

We run a further set of experiments to gauge the performance of TSSP-ILS using an extended runtime. When the runtime is doubled to $2\,T$ (380s), we are able to find feasible solutions for 19 of the 20 instances as shown in Table 5.27. In fact, we managed to obtain best known results for all the instances. In addition, we conducted a t-test to compare the means of TSSP-ILS with runtime of T and $2\,T$. The p values (less than 0.05) indicates a significant difference between the means of TSSP-ILS for both runtimes. We further extended the runtime for

Hard big instances with non zero means (B6, B7, B17, B18, B19 and B20). The results are illustrated in Figures 5.1-5.6, meanwhile the respective descriptive statistics are given in Tables 5.28-5.33. Note that the circles and stars in the box plots are mild and extreme outliers.

	runtime=T			*runtime*=$2T$			
		Unassigned			Unassigned		t-test
Instance	Fea.(%)	best	mean	Fea.(%)	best	mean	(p value)
B1	100	0	0.00	100	0	0.00	-
B2	100	0	0.00	100	0	0.00	-
B3	100	0	0.00	100	0	0.00	-
B4	100	0	0.00	100	0	0.00	-
B5	100	0	0.00	100	0	0.00	-
B6	16	0	1.87	35	0	1.00	0.006
B7	100	23	29.55	100	21	26.87	0.006
B8	100	0	0.00	100	0	0.00	-
B9	100	0	0.00	100	0	0.00	-
B10	100	0	0.00	100	0	0.00	-
B11	100	0	0.00	100	0	0.00	-
B12	100	0	0.00	100	0	0.00	-
B13	100	0	0.00	100	0	0.00	-
B14	100	0	0.00	100	0	0.00	-
B15	100	0	0.00	100	0	0.00	-
B16	100	0	0.00	100	0	0.00	-
B17	6	0	3.97	19	0	1.45	0.000
B18	48	0	0.84	81	0	0.23	0.004
B19	0	1	16.97	3	0	9.16	0.000
B20	0	2	6.65	6	0	3.35	0.000

Table 5.27: Comparison between TSSP-ILS with runtime of T and $2T$ for Hard big instances. n=31 runs.

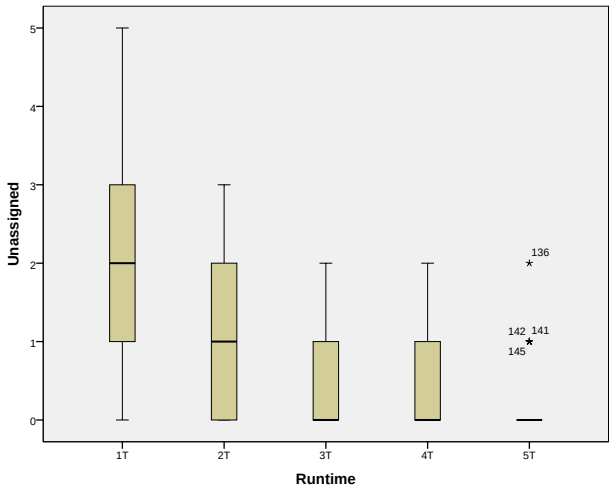

Figure 5.1: Box plot showing the number of unassigned events for TSSP-ILS with an extended runtime on Hard-B6 instance. n=31 runs.

	Runtime				
Unassigned	1T	2T	3T	4T	5T
Min	0	0	0	0	0
Max	5	3	2	2	2
Median	2.00	1.00	0.00	0.00	0.00
Mean	1.87	1.00	0.39	0.35	0.26

Table 5.28: Descriptive statistics for TSSP-ILS with an extended runtime on Hard-B-6 instance. n=31 runs.

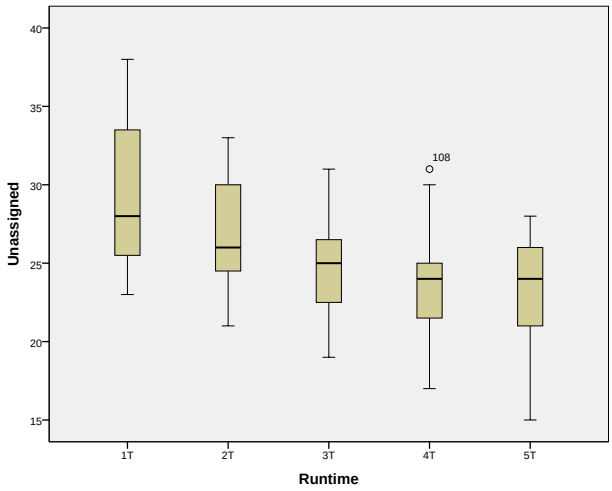

Figure 5.2: Box plot showing the number of unassigned events for TSSP-ILS with an extended runtime on Hard-B7 instance. n=31 runs.

	Runtime				
Unassigned	1T	2T	3T	4T	5T
Min	23	21	19	17	15
Max	38	33	31	31	28
Median	28.00	26.00	25.00	24.00	24.00
Mean	29.55	26.87	24.58	23.74	23.13

Table 5.29: Descriptive statistics for TSSP-ILS with an extended runtime on Hard-B7 instance. n=31 runs.

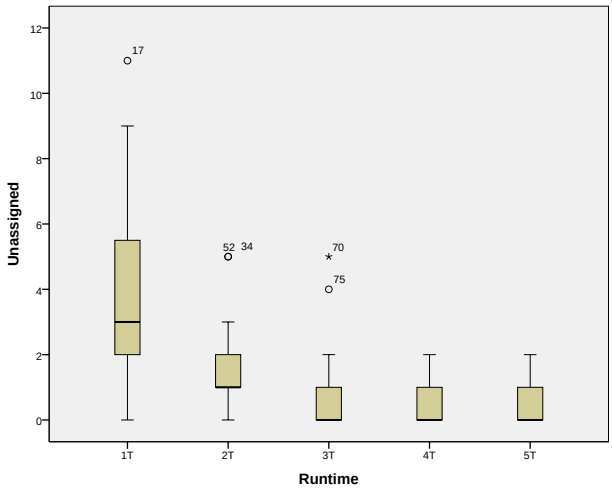

Figure 5.3: Box plot showing the number of unassigned events for TSSP-ILS with an extended runtime on Hard-B17 instance. n=31 runs.

	Runtime				
Unassigned	1T	2T	3T	4T	5T
Min	0	0	0	0	0
Max	11	5	5	2	2
Median	3.00	1.00	0.00	0.00	0.00
Mean	3.97	1.45	0.71	0.61	0.32

Table 5.30: Descriptive statistics for TSSP-ILS with an extended runtime on Hard-B17 instance. n=31 runs.

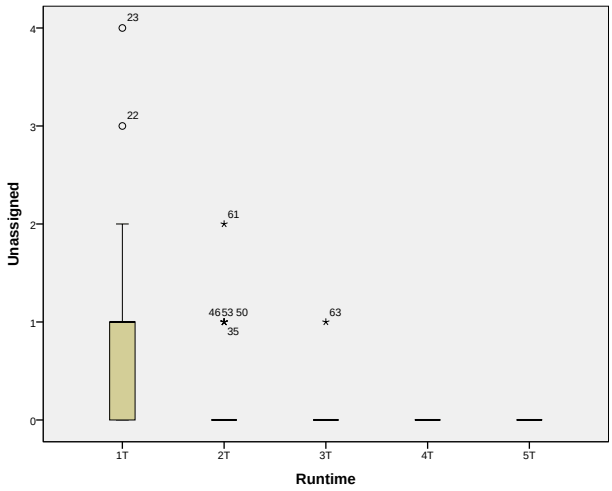

Figure 5.4: Box plot showing the number of unassigned events for TSSP-ILS with an extended runtime on Hard-B18 instance. n=31 runs.

	Runtime				
Unassigned	1T	2T	3T	4T	5T
Min	0	0	0	0	0
Max	4	2	1	0	0
Median	1.00	0.00	0.00	0.00	0.00
Mean	0.84	0.23	0.03	0.00	0.00

Table 5.31: Descriptive statistics for TSSP-ILS with an extended runtime on Hard-B18 instance. n=31 runs.

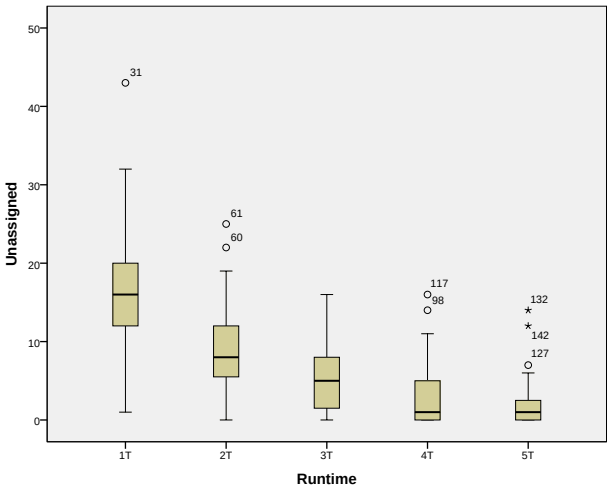

Figure 5.5: Box plot showing the number of unassigned events for TSSP-ILS with an extended runtime on Hard-B19 instance. n=31 runs.

	Runtime				
Unassigned	1T	2T	3T	4T	5T
Min	1	0	0	0	0
Max	43	25	16	16	14
Median	16.00	8.00	5.00	1.00	1.00
Mean	16.97	9.16	5.19	3.39	2.06

Table 5.32: Descriptive statistics for TSSP-ILS with an extended runtime on Hard-B19 instance. n=31 runs.

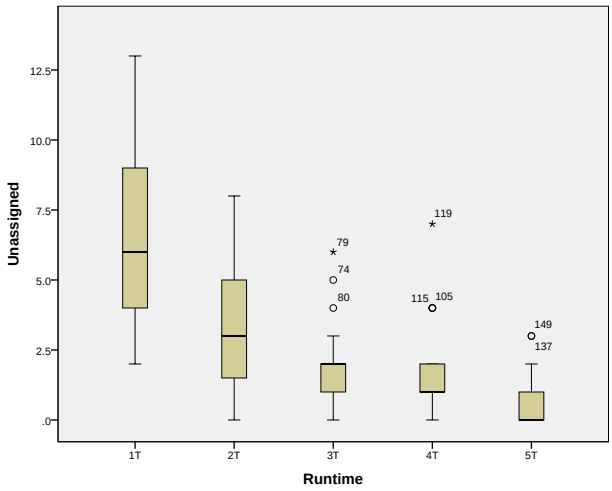

Figure 5.6: Box plot showing the number of unassigned events for TSSP-ILS with an extended runtime on Hard-B20 instance. n=31 runs.

	Runtime				
Unassigned	1T	2T	3T	4T	5T
Min	2	0	0	0	0
Max	13	8	6	7	3
Median	3.00	3.00	2.00	1.00	0.00
Mean	6.65	3.35	1.90	1.45	0.68

Table 5.33: Descriptive statistics for TSSP-ILS with an extended runtime on Hard-B20 instance. n=31 runs.

5.3 Discussion

TSSP does not require parameter tuning as the values such as event sampling size S and $ITER$ in the algorithm are determined automatically based on the characteristics of the specific instances.

TSSP is not only effective but also fast in finding feasible solutions. In our opinion, the sampling of events reduces the number of evaluations needed before a move is made, permitting more moves per time unit.

In our implementation, the sampling ratio (event sampling size : number

of unassigned events) increases as more events are assigned. For instance, the event sampling size for 1000 events is 3 (0.0025 $\times$ 1000 or 0.25% of the # of events). The event sampling size is fixed throughout the search. Initially, the sampling ratio is 0.0025 (0.25%). When the number of unassigned events decreases to 6, the sampling ratio is 0.5 (50%). When the number of unassigned events decreases to 3 or below, the sampling ratio is 1.0 (100%) where all the events will be selected for evaluation without sampling. Naturally, the event sampling that we adopted allows the search to switch from diversification (exploration) to intensification (exploitation) and vice versa depending on the number of unassigned events.

Meanwhile, the proposed cost function increases the probability of unassigned events to be assigned later as they have the least number of clash with other events.

The perturbation that we proposed attempts to move all assigned events to random time slots instead of trying to move random assigned events to random time slots. As a result, a solution is thoroughly perturbed. Ideally, perturbation should be initiated when cycling (local optima) is detected. Some researchers perturb the solution after certain idle iterations. However, the solution may be perturbed prematurely as the search may not be stuck and still looking for good solution especially when the search space is large. We perturb the solution after certain intervals proportional to the size of search space. This way, we hope that the search will be able to examine the current area of the search space thoroughly before perturbation is initiated. After perturbation, the search explores other areas of the search space or possibly escapes from local optima (if the search is stuck). The importance of perturbation is shown in Figures 5.7 and 5.8. Without perturbation, the search is stuck when cycling occurs (indicated by idle current cost). As a result, there is one event remains unassigned. On the contrary, when perturbation is enabled, TSSP found a feasible solution in an equally seeded run.

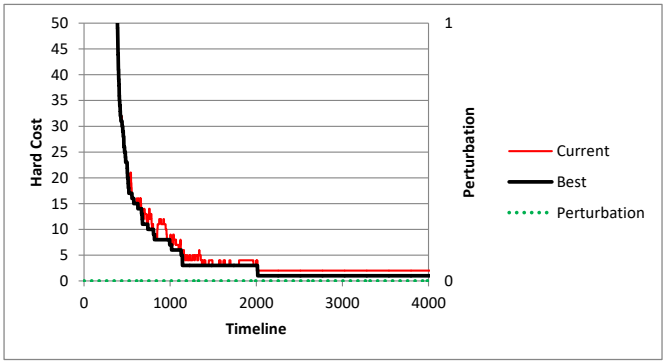

Figure 5.7: TSSP (perturbation disabled) on Hard-M15. The number of unassigned event is 1.

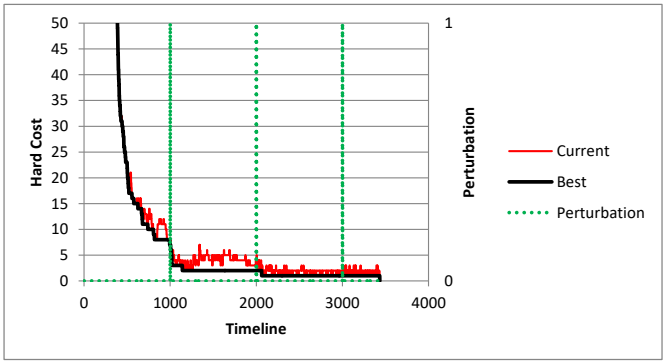

Figure 5.8: TSSP on Hard-M15. The number of unassigned event is 0.

TSSP is good in both exploring and exploiting the search space. The best solution is further improved by ILS towards the end of the search. ILS guides the search to operate in the vicinity of the best solution hoping to find the optimal solution. The local search (first descent) that we employ in ILS is not only computationally fast, but also exploitative as it only accepts improving or equal cost solutions. In addition, two operators are used randomly in ILS namely transfer and swap to ensure the solution space is well connected.

5.4 Conclusion

We have presented the effect of sampling on TS. TS with event sampling is more effective than TS without event sampling. Event sampling size S=[0.25% $\times$ # of events], is more effective than event sampling size $S = 1$, particularly when the # of events is high. Event sampling size S=[0.25% $\times$ # of events] not only allows more moves per time unit but also naturally varies the diversification and intensification capability of the algorithm (depending on the the number of unassigned events) despite the fixed S throughout the search.

We compared the effect of using different cost functions with or without perturbation on TS with sampling. The novel cost function proposed in Eq. 5.1 performed better than other cost functions used in the scientific literature regardless of whether perturbation is used.

The results are further improved when the proposed perturbation is paired with the proposed cost function on TS with event sampling. The proposed cost function is exploitative, and the proposed perturbation is exploratory. A good balance between exploration (proposed perturbation) and exploitation (proposed cost function) for the search is achieved when the perturbation is initiated at the right time. When perturbation is called too early (more frequent), the search lacks exploitation. Meanwhile if perturbation is initiated too late (less frequent), the search lacks exploration.

Overall, TSSP is shown to be more effective in finding feasible solutions for the benchmark timetabling problem compared to TS. The number of unassigned events and average time to feasibility are presented for all the datasets. In addition, t-tests are conducted to compare the means for these values between TS and TSSP. TSSP managed to find 100% feasibility for all Socha, ITC02 and ITC07 instances in relatively short time compared to existing methods in the scientific literature.

TSSP is further enhanced by hybridization with ILS. We compared the effectiveness of TSSP and TSSP-ILS in finding feasible solutions for considerably hard benchmarked timetabling problems. The method is not only superior to the performance of either TSSP or ILS alone but also other state of the art

methods. In addition, significant improvement of result is observed when the runtime is extended suggesting that TSSP-ILS is extendable. As we have developed an efficient algorithm in finding feasible solutions, we shift our focus on improving the soft constraint violations of the feasible solutions in the next chapter.

Chapter 6

Improving Quality: SAR Algorithm

In this chapter, we improve the feasible solutions in terms of soft constraint violations by using a method based on SA. We do not use Tabu Search (TS) to improve the soft constraint violations as we feel TS is too restrictive and may affect the connectivity of search space. Instead, we focus on SA as it has been very effective in solving combinatorial optimization problems, particularly timetabling problems. In fact, all state of the art methods for the instances considered in this work are based on SA. The work in this chapter has been published as [88]:

- Say Leng Goh, Graham Kendall, Nasser R. Sabar. Improved Local Search Approaches to Solve Post Enrolment Course Timetabling Problem. European Journal of Operational Research, 2017.

6.1 SA with Reheating (SAR)

SA generally accepts all improving and equal cost candidate solutions and accepts worsening candidate solutions with certain probabilities depending on the current temperature. This property of accepting worsening candidate solutions enables the search to escape from local optima.

The drawback of SA is the requirement to tune a number of parameters sometimes for a set of benchmark instances and sometimes for specific instances, in

order to get competitive results. Generally, the parameters which require tuning are the initial temperature, end temperature, Markov chain length and cooling rate. Usually, preliminary runs are initiated to gauge the suitable initial temperature (based on standard deviation of the cost) [178].

In geometric cooling, as the temperature gradually decreases, the search gradually switches from exploring to exploiting the search space. At relatively high temperatures, the current solution may stray away from the best solution. Meanwhile, at relatively low temperatures, the search may stall (probably trapped when most of the worse transitions will be rejected). In both scenarios, we may observe idle iterations, where no new best is achieved. Some researchers rely on the idle iteration count to estimate whether the search is stuck in local optima before triggering any exploration initiative such as reheating [6] or channeling the search process to other algorithms [134]. Reheating the temperature after certain idle iterations may not be the best option as it may result in unnecessary (premature) reheating and affect the exploitation capability of SA.

The algorithm presented below is developed with consideration on these aspects. We suggest solutions on issues such as the ideal time to reheat and the level of reheating required in order for the search to escape from local optima.

6.1.1 Algorithm Description

We are proposing an improved Simulated Annealing with Reheating (SAR). The method is inspired by the idea that when the current cost is high, the search should explore more and when the current cost is low, the search should exploit more. In SAR, we rely on the current cost to determine the initial temperature (rigorous setting of the initial temperature is bypassed) and how much to reheat when the search is stuck. In fact, we also rely on the current cost to determine whether the search is stuck in a local optima (inactive current cost through Markov chains indicates the search is stuck). As the temperature is reheated when a local optima is estimated at a certain low temperature, the setting of an end temperature as required in conventional SA is omitted. If the search is still stuck after the previous reheating, a higher temperature is applied for the next reheating. We estimate whether the search is still stuck in the previous local optima by utilising the current and best cost. The approach is novel as the closest cost based reheating in the literature is based on the best cost and

specific heat [10, 72]. The details of SAR is shown in Algorithm 36.

Algorithm 36

```
1: procedure SAR(current, E)
2:     temp ← f(current) × C
3:     heat ← 0
4:     best ← current
5:     previousCost ← f(current)
6:     currentStagnantCount ← 0
7:     stuckedBestCost ← f(current)
8:     stuckedCurrentCost ← f(current)
9:
10:    while current is not optimal AND time.elapsed() < runtime do
11:        for all e ∈ E do
12:            moved ← false
13:            for slot = 1 to 45 do
14:                ns ← SELECTNEIGHBOURSTRUCTURE()
15:                candidate ← GETCANDIDATE(current, e, slot, ns)
16:                if candidate exists then
17:                    if RANDOM[0,1) ≤ exp(−(f(candidate)−f(current))/temp) then
18:                        moved ← true
19:                        current ← candidate
20:                        if f(current) < f(best) then
21:                            best ← current
22:                        end if
23:                    end if
24:                end if
25:                if moved then
26:                    break
27:                end if
28:            end for
29:        end for
```

```
30:         if STUCK(f(current), previousCost, currentStagnantCount) then
31:             if f(best) = stuckBestCost then
32:                 if f(current) − stuckCurrentCost < 2% then
33:                     heat = heat + 1
34:                 else
35:                     heat ← 0
36:                 end if
37:             else
38:                 heat ← 0
39:             end if
40:             temp ← [heat × 0.2 × f(current) + f(current)] × C
41:             stuckBestCost ← f(best)
42:             stuckCurrentCost ← f(current)
43:         else
44:             temp ← temp × β
45:         end if
46:         previousCost ← f(current)
47:     end while
48: end procedure
```

Algorithm 37

```
1: procedure STUCK(f(current), previousCost, currentStagnantCount)
2:     if f(current) − previousCost < 1% then
3:         currentStagnantCount = currentStagnantCount + 1
4:     else
5:         currentStagnantCount ← 0
6:     end if
7:     if currentStagnantCount > THRESHOLD then
8:         return true
9:     else
10:        return false
11:    end if
12: end procedure
```

Algorithm 38

```
1: procedure SELECTNEIGHBOURSTRUCTURE( )
2:     return a neighbourhood structure selected probalistically (Roulette
   Wheel) from a set of neighbourhood structures with predefined composi-
   tion.
3: end procedure
```

At each temperature, a Marcov chain is generated by deterministically try-

ing to move each event $e \in E$ into each time slot (except the time slot currently occupied by e) using a neighbourhood structure selected probalistically from a set of neighbourhood structures with predefined composition as shown in Table 6.1. Note that the composition is empirically adjusted. We use maximal matching (only when necessary) for room assignment.

Candidate solutions are feasible solutions which satisfy all the hard constraints. If a candidate solution exists, it is evaluated using the acceptance criterion where the improving and equal cost solution is accepted while the worsening solution is accepted with a certain probability. If accepted, the candidate solution will be set as the current solution. If the current solution is better than the best, the best solution is updated.

Dataset	Neighbourhood Structure Composition (%)
Socha	Transfer: 70, Swap: 29, Kempe: 1
ITC02	Transfer: 70, Swap: 29, Kempe: 1
ITC07	Transfer: 70, Swap: 20, Kempe: 10

Table 6.1: Neighbourhood structure composition for dataset

The neighborhood structures used are:

- Transfer: Attempt to transfer e into *slot*. A feasible transfer is returned as a candidate for acceptance evaluation.
- Swap: A swap is attempted between e with event in each room (incrementing order) in *slot*. The first feasible swap is returned as a candidate for acceptance evaluation.
- Kempe: Same as described in section 5.1 except that a candidate is returned for acceptance evaluation.

In our implementation, a number of variables are maintained, namely *previousCost* (cost after each Markov chain is completed), *currentStagnantCount* (number of consecutive times current cost remains the same), *stuckBestCost* (best cost when the search is stuck) and *stuckCurrentCost* (current cost when the search is stuck).

The initial temperature is set as the initial cost multiplied by a constant C. This bypasses the manual setting of an initial temperature which is critical in conventional SA. The temperature is cooled according to an update rule

$T_{i+1} = T_i \times \beta$.

After each Marcov chain, we check whether the search is stuck in a local optima. In the STUCK procedure (Algorithm 37), we observe the changes of the current cost between Markov chains. *currentStagnantCount* is incremented by 1 if the difference between *f(current)* and *previousCost* is less than 1%. Otherwise, *currentStagnantCount* is set to 0.

If the search is stuck (*currentStagnantCount* is more than a THRESHOLD value set as 5), then the temperature is reheated according to

$$temp \leftarrow [heat \times 0.2 \times f(current) + f(current)] \times C \tag{6.1}$$

where C is a constant and *heat* is the incremental step. For the first reheating, *heat* is usually set to 0 (line 38) thus the temperature is reheated to $f(current) \times C$ before being cooled again until the search is stuck in another local optima.

If the search is still stuck in the previous local optima (no new best since the previous reheating AND $f(current) - stuckCurrentCost < 2\%$), then *heat* is incremented by 1 (line 33). Essentially, a higher temperature is applied for the next reheating so that the search can explore more in order to escape from the previous local optima.

If the search has escaped from the previous local optima ([a new best since the previous reheating] OR [no new best since the previous reheating AND $f(current) - stuckCurrentCost \geq 2\%$]), then *heat* is set to 0 (line 35 and 38). In effect, the temperature is reheated to $f(current) \times C$ in order for the search to escape from the current local optima. Note that the setting of end temperature is omitted as the temperature is reheated when the search is stuck.

The series of cooling and reheating is repeated until an optimal solution is obtained or the elapsed time exceeds *runtime*. We set the decay rate β to 0.9995 and the constant C to 0.01. The same settings are used across all instances in our experiments.

6.1.2 Experimental Results

6.1.2.1 Comparing Neighborhood Structures

To compare the effect of neighborhood structures, the temperature is cooled statically without reheating from the initial value until the elapsed time exceeds the time limit.

In each Marcov chain, we attempt to move a certain number (instance specific) of randomly selected events. To move each selected event, up to 45 (number of time slots) randomly selected time slots are attempted, by using a neighborhood operator which is selected probabilistically from a predefined composition. Therefore, the number of attempts in each Marcov chain ranges from # of events to # of events $\times$ 45. The number of attempts varies as some events are moved to the first selected time slot while others are not moved even after the 45th time slot. The next event is immediately considered when the present event is successfully moved.

The neighborhood structures tested separately are transfer, swap and kempe. We also tested the mixture of the neighborhood structures at ratio of 33:33:33 and 70:29:1 for the transfer: swap: kempe operators. The number of soft constraint violations, trial moves, feasible moves and accepted moves for the neighborhood structures are shown in Tables 6.2, 6.3 and 6.4 for instance Socha-L, ITC02-1 and ITC07-1. The feasible ratio and accepted ratio are calculated by Feasible/Trial and Accepted/Trial respectively. Based on the trial number, transfer operator is the fastest (lowest computational cost), followed by swap and kempe operator. Kempe operator has the highest value for both the feasible ratio and accepted ratio. Based on the soft cost, the ratio 70:29:1 works best for Socha-L and ITC02-1 instances. Meanwhile, 33:33:33 is the most effective ratio for instance ITC07-1.

	Neighborhood Structure				
Variable	Transfer	Swap	Kempe	33:33:33	70:29:1
Soft Cost	448.52	463.23	1013.39	940.13	218.42
Trial (K)	627236	252377	3539	9843	188926
Feasible (K)	28637	14006	1847	1924	10431
Accepted (K)	1238	1455	583	615	1148
Feasible Ratio	0.0461	0.0556	0.5218	0.1955	0.0553
Accepted Ratio	0.0020	0.0058	0.1648	0.0625	0.0061

Table 6.2: Comparison among neighborhood structure on Socha-L instance. n=31 runs.

	Neighborhood Structure				
Variable	Transfer	Swap	Kempe	33-33-33	70-29-1
Soft Cost	111.10	226.03	492.45	390.55	60.84
Trial (K)	636770	244990	11200	29785	305461
Feasible (K)	49064	8434	2578	2863	20269
Accepted (K)	590	919	824	845	716
Feasible Ratio	0.0771	0.0345	0.2302	0.0961	0.0664
Accepted Ratio	0.0009	0.0038	0.0736	0.0284	0.0023

Table 6.3: Comparison among neighborhood structure on ITC02-1 instance. n=31 runs.

	Neighborhood Structure				
Variable	Transfer	Swap	Kempe	33-33-33	70-29-1
Soft Cost	1706.29	2644.42	430.00	358.23	956.90
Trial (K)	1617666	1158748	70061	132516	700448
Feasible (K)	15935	2095	1266	3439	11985
Accepted (K)	477	153	369	355	345
Feasible Ratio	0.0102	0.0018	0.0181	0.0263	0.0180
Accepted Ratio	0.0003	0.0001	0.0053	0.0027	0.0005

Table 6.4: Comparison among neighborhood structure on ITC07-1 instance. n=31 runs.

The soft constraint violations for the selected instances are summarized in Table 6.5. As the ratio 70:29:1 has the lowest means for 2 out of 3 selected instances, it is therefore used from here on.

	Neighborhood Structure									
	Transfer		Swap		Kempe		33-33-33		70-29-1	
Instance	best	mean	best	mean	best	mean	best	mean	best	mean
Socha-L	371	448.52	376	463.23	968	1013.39	882	940.13	169	**218.42**
ITC02-1	96	111.10	203	226.03	465	492.45	367	390.55	48	**60.84**
ITC07-1	1251	1706.29	2222	2644.42	129	430.00	0	**358.23**	533	956.90
Avg.	-	755.30	-	1111.23	-	645.28	-	562.97	-	**412.05**

Table 6.5: Comparison of soft constraint violations among neighborhood structure on selected instances. n=31 runs.

6.1.2.2 Comparing Neighborhood Examination Scheme

As in the previous section, the temperature is decreased statically without reheating from the initial value until the time limit is exceeded. The neighborhood structure composition is fixed at 70:29:1 for the transfer:swap:kempe operators. In this section, we compare several methods of selecting events and time slots, namely RE-RS (random event and random slot), DE-RS (deterministic event and random slot) and DE-DS (deterministic event and deterministic slot). We note that RE-RS was used in the previous section (6.1.2.1). The best and mean of soft constraint violations for selected instances are given in Table 6.6. DE-DS has the lowest average of means for all the selected instances. As DE-DS is the most effective among the neighborhood examination schemes in terms of the number of instances with the lowest mean, it is used from here on.

	Neighborhood Examination Scheme					
	RE-RS		DE-RS		DE-DS	
Instance	best	mean	best	mean	best	mean
Socha-L	169	218.42	174	**207.58**	168	210.39
ITC02-1	48	60.84	34	59.32	37	**52.10**
ITC07-1	533	956.90	396	832.87	420	**800.74**
Avg.	-	412.05	-	366.59	-	**354.41**

Table 6.6: Comparison of soft constraint violations among neighborhood examination schemes on selected instances. n=31 runs.

6.1.2.3 The Effect of Basic Reheating

Here, we investigate the effect of basic reheating. The temperature is decremented statically from the initial value and reheated when the search is stuck. The search is assumed stuck if the current cost changes between Marcov chains is nil for a given number of consecutive times (*THRESHOLD* value is set as 5). The temperature is reheated to a value as a function of the current cost. The series of cooling and reheating are repeated until the time limit is exceeded. The neighborhood structure composition is fixed at 70:29:1 ratio and DE-DS is used as the neighborhood examination scheme. As evident in Table 6.7, the means for all the selected instances are further improved when basic reheating is used. Therefore, basic reheating is used from now on.

	Basic Reheating			
	disabled		enabled	
Instance	best	mean	best	mean
Socha-L	168	210.39	146	**191.03**
ITC02-1	37	52.10	22	**32.97**
ITC07-1	420	800.74	72	**549.10**
Avg.	-	354.41	-	**257.70**

Table 6.7: The effect of basic reheating upon soft constraint violations on selected instances. n=31 runs.

We tested several values for C as given in Table 6.8. The value of 0.01 recorded the lowest means for 2 out of the 3 selected instances. Therefore, it is used onwards.

	Values of C					
	0.005		0.01		0.02	
Instance	best	mean	best	mean	best	mean
Socha-L	175	207.35	146	**191.03**	158	203.65
ITC02-1	35	46.81	22	**32.97**	35	45.65
ITC07-1	553	952.26	72	549.10	0	**325.19**
Avg.	-	402.14	-	257.70	0	**191.49**

Table 6.8: The results of setting different values for C upon soft constraint violations on selected instances. n=31 runs.

We also tested a couple of values for $THRESHOLD$ as given in Table 6.9. The value of 5 recorded the lowest means for all the 3 selected instances. Thus, it is used onwards.

	Values of *THRESHOLD*			
	5		10	
Instance	best	mean	best	mean
Socha-L	146	**191.03**	153	196.94
ITC02-1	22	**32.97**	27	35.68
ITC07-1	72	**549.10**	282	581.48
Avg.	-	**257.70**	-	271.37

Table 6.9: The results of setting different values for *THRESHOLD* upon soft constraint violations on selected instances. n=31 runs.

6.1.2.4 Comparing Local Optima Detection

To determine whether the search is stuck, we keep track of the current cost changes between Marcov chains. If the current cost is stagnant for a given number of consecutive times (set as 5), the search is considered stuck. The current cost is perceived as stagnant if changes is nil (Type 1) or the changes is less than 1% (Type 2) or the changes is less than 2% (Type 3). Type 1 was used in the previous section (6.1.2.3). Table 6.10 shows that Type 2 is more effective than Type 1 and Type 3 in terms of number of instances with the lowest means for the selected instances. Note that the neighborhood structure composition is fixed at 70:29:1 ratio and DE-DS is used as the neighborhood examination scheme.

	Local Optima Detection					
	Type 1		Type 2		Type 3	
Instance	best	mean	best	mean	best	mean
Socha-L	146	**191.03**	143	199.03	271	613.03
ITC02-1	22	32.97	24	**32.90**	48	151.87
ITC07-1	72	549.10	129	**332.65**	0	575.19
Avg.	-	257.70	-	**188.19**		(446.70)

Table 6.10: Comparison of soft constraint violations between local optima detection type 1,2 and 3 on selected instances. n=31 runs.

6.1.2.5 The Effect of Incremental Reheating

Incremental reheating allows the temperature to be reheated to a higher level if the search is still stuck since the previous reheating. We compare the performance of the algorithm with basic and incremental reheating. Note that the neighborhood structure composition is fixed at 70:29:1 ratio, DE-DS is used as the neighborhood examination scheme and local optima detection Type 2 is utilized. Incremental reheating is used onwards as it recorded the lowest means for 2 out of 3 selected instances as shown in the Table 6.11.

	Reheating			
	Basic		Incremental	
Instance	best	mean	best	mean
Socha-L	143	**199.03**	165	206.6
ITC02-1	24	32.90	22	**32.29**
ITC07-1	129	332.65	0	**317.32**
Avg.	-	188.19	-	**183.06**

Table 6.11: Comparison of soft constraint violations between basic and incremental reheating on selected instances. n=31 runs.

We tested several values for β as given in Table 6.12. One lowest mean is recorded by each value of β. However, 0.9995 is the most effective in terms of average of means for the selected instances. Therefore, this value is used onwards.

	Values of β					
	0.999		0.9995		0.9999	
Instance	best	mean	best	mean	best	mean
Socha-L	148	**202.00**	165	206.6	518	632.45
ITC02-1	24	34.23	22	**32.29**	128	174.52
ITC07-1	127	391.65	0	317.32	9	**191.52**
Avg.	-	209.29	-	**183.06**	-	332.83

Table 6.12: The results of setting different values for β upon soft constraint violations on selected instances. n=31 runs.

6.1.2.6 Comparing SAR with State of the Art Methods

We now compare SAR with the best results in the literature. Table 6.13 summarizes the details of the solvers.

Solver	Technique	Reference
A	Ant System	Socha et al. [164]
B	Tabu Search Hyperheuristic	Burke et al. [35]
C	Extended Great Deluge	McMullan [129]
D	Great Deluge + Tabu Search	Abdullah et al. [7]
E	Non Linear Great Deluge + Learning	Obit et al. [136]
F	Fish Swarm	Turabieh et al. [176]
G	Round Robin Multi Algorithms	Shaker and Abdullah [161]
H	Honey Bee Mating	Sabar et al. [155]
I	Simulated Annealing	Ceschia et al. [48]
J1	Simulated Annealing	Kostuch [109]
J2	Simulated Annealing	Kostuch [110]
K	Tabu Search	Cordeau et al. [55]
L	Great Deluge	Burke et al. [27]
M	Local Search + Tabu Search	DiGaspero and Schaerf [63]
N	Hybrid Algorithm	Chiarandini et al. [52]
O	Simulated Annealing	Cambazard et al. [41]
P	Ant Colony Optimization	Nothegger et al. [135]
Q	Simulated Annealing	Lewis and Thompson [118]

Table 6.13: Solver details

SAR outperformed (best results are in bold) all the other solvers for all Socha instances as shown in Table 6.14. It is interesting to note that our averages are far better than the best produced by other solvers over all instances. We found optimal solutions for 9 out of the 11 instances. Note that solver A was run according to the time limit set initially (small instances: 90s, medium instances: 900s and large instances: 9000s). For solver B-H, no benchmark time limit was followed in their implementations.

Instance	Solver									
	A	B	C	D	E	F	G	H	I	SAR
S1	1	1	**0**(0.8)	**0**	**0**	**0**	**0**	**0**	**0**(**0.0**)	**0**(**0.0**)
S2	3	2	**0**(2.0)	**0**	**0**	**0**	**0**	**0**	**0**(**0.0**)	**0**(**0.0**)
S3	1	**0**	**0**(1.3)	**0**	**0**	**0**	**0**	**0**	**0**(**0.0**)	**0**(**0.0**)
S4	1	1	**0**(1.0)	**0**	**0**	**0**	**0**	**0**	**0**(0.1)	**0**(**0.0**)
S5	**0**	**0**	**0**(0.2)	**0**	**0**	**0**	**0**	**0**	**0**(**0.0**)	**0**(**0.0**)
M1	195	146	80(101.4)	78	38	45	117	75	9(26.5)	**0**(**1.5**)
M2	184	173	105(116.9)	92	37	40	108	88	15(25.9)	**0**(**2.2**)
M3	248	267	139(162.1)	135	60	61	135	129	36(49.0)	**7**(**13.4**)
M4	164.5	169	88(108.8)	75	39	35	75	74	12(23.8)	**0**(**0.7**)
M5	219.5	303	88(119.7)	68	55	49	160	64	3(10.9)	**0**(**1.2**)
L	851.1	1166	730(834.1)	556	638	407	589	523	208(259.8)	**165**(**206.6**)

Table 6.14: Comparing SAR with other solvers on Socha instances. Depicted is best(mean) of soft constraint violations. n=31 runs. Note that some authors only reported their best results.

Results comparison for ITC02 is given in Table 6.15. J1 was the official winner of ITC02. The solver N appeared post competition and was competitive with the solver J1. Not long after that, J2 was presented and became the state of the art method with the best known results for all the instances. J2 was an improvement of J1 by the same author. Since then, no other solvers are able to beat the results of J2 on any of the ITC02 instances. We have managed to achieve that. Our results are competitive, or better, than the other solvers on all the instances. In fact, we managed to get optimal solutions for 7 out of 20 instances in comparison to the four of J2.

	Solver							
Instance	J1	K	L	M	N	J2	I	SAR
1	45	61	85	63	45	**16**(**30.2**)	45(57.1)	23(32.6)
2	25	39	42	46	14	**2**(**11.4**)	20(33.2)	7(13.7)
3	65	77	84	96	45	**17**(**31.0**)	43(53.2)	26(36.4)
4	115	160	119	166	71	**34**(**60.8**)	87(109.9)	50(63.1)
5	102	161	77	203	59	42(72.1)	71(91.7)	**38**(**58.6**)
6	13	42	6	92	1	**0**(2.4)	2(14.1)	**0**(**0.8**)
7	44	52	12	118	3	2(8.9)	2(13.7)	**0**(**2.6**)
8	29	54	32	66	1	**0**(2.0)	9(20.0)	**0**(**1.4**)
9	17	50	184	51	8	1(5.8)	15(21.9)	**0**(**4.6**)
10	61	72	90	81	52	**21**(**35.0**)	41(60.7)	28(40.9)
11	44	53	73	65	30	**5**(**12.9**)	24(38.2)	10(17.7)
12	107	110	79	119	75	55(76.3)	62(83.7)	**53**(**64.5**)
13	78	109	91	160	55	**31**(**47.1**)	59(78.0)	38(53.3)
14	52	93	36	197	18	11(22.3)	21(34.2)	**5**(**12.9**)
15	24	62	27	114	8	2(8.4)	6(11.8)	**0**(**4.0**)
16	22	34	300	38	55	**0**(3.4)	6(16.7)	**0**(**0.5**)
17	86	114	79	212	46	37(54.0)	42(56.5)	**26**(**41.6**)
18	31	38	39	40	24	4(**9.4**)	11(25.9)	**2**(9.7)
19	44	128	86	185	33	**7**(**16.4**)	56(73.0)	11(24.7)
20	7	26	**0**	17	**0**	**0**(0.5)	**0**(1.8)	**0**(**0.0**)

Table 6.15: Comparing SAR with other solvers on ITC02 instances. Depicted is best(mean) of soft constraint violations. n=31 runs. Note that some authors only reported their best results.

Table 6.16 shows the results comparison for ITC07. The solver O was the official winner of ITC07. The solver P is based on Ant Colony Optimization and is the only competitve algorithm which is not SA based. However, SA was present in their approach and played a critical role in its performance. Our results are competitive compared to the other solvers. We managed to obtain

optimal solutions for 15 out of 24 instances. Solvers O and P did not attempt their methods on instances 17-24.

Instance	Solver				
	O	P	I	Q	SAR
1	15(547.0)	**0**(613.0)	59(399.2)	**0**(377.0)	**0**(**307.6**)
2	9(403.0)	**0**(556.0)	**0**(142.2)	**0**(382.2)	**0**(**63.4**)
3	174(254.0)	**110**(680.0)	148(209.9)	122(**181.8**)	163(199.4)
4	249(361.0)	53(580.0)	25(349.6)	**18**(**319.4**)	242(328.8)
5	**0**(26.0)	13(92.0)	**0**(7.7)	**0**(7.5)	**0**(**2.7**)
6	**0**(16.0)	**0**(212.0)	**0**(**8.6**)	**0**(22.8)	**0**(33.2)
7	1(8.0)	**0**(4.0)	**0**(4.9)	**0**(5.5)	5(18.0)
8	**0**(**0.0**)	**0**(61.0)	**0**(1.5)	**0**(0.6)	**0**(**0.0**)
9	29(1167.0)	**0**(202.0)	**0**(258.8)	**0**(514.4)	**0**(**100.7**)
10	2(1297.0)	**0**(**4.0**)	3(186.4)	**0**(1202.4)	**0**(65.3)
11	178(361.0)	143(774.0)	142(269.5)	**48**(**202.6**)	161(244.3)
12	14(380.0)	**0**(538.0)	267(400.0)	**0**(340.2)	**0**(**318.2**)
13	**0**(135.0)	5(360.0)	1(120.0)	**0**(79.0)	**0**(99.5)
14	**0**(15.0)	**0**(41.0)	**0**(3.6)	**0**(0.5)	**0**(**0.2**)
15	**0**(47.0)	**0**(**29.0**)	**0**(48.0)	**0**(139.9)	**0**(192.0)
16	1(58.0)	**0**(101.0)	**0**(**50.1**)	**0**(105.2)	10(105.8)
17	-	-	**0**(**0**)	**0**(0.1)	**0**(0.8)
18	-	-	**0**(41.1)	**0**(**2.2**)	**0**(12.5)
19	-	-	**0**(951.5)	**0**(**346.1**)	**0**(516.7)
20	-	-	543(700.2)	557(724.5)	586(**650.7**)
21	-	-	5(35.9)	1(32.1)	**0**(**12.5**)
22	-	-	5(**19.9**)	4(1790.1)	**1**(136.0)
23	-	-	1292(1707.7)	**0**(514.1)	11(**504.4**)
24	-	-	**0**(**105.3**)	18(328.2)	5(192.6)

Table 6.16: Comparing SAR with other solvers on ITC07 instances. Depicted is best(mean) of soft constraint violations. n=31 runs.

6.1.2.7 Extended Runtime for SAR

Up to this point, all the experiments were conducted according to the time limit provided by the competition benchmark program. Out of curiosity, we also performed some experiments to see the effects of an extended runtime with regard to soft constraint violation. We selected one hard instance from each dataset namely Socha-L, ITC02-1, ITC07-1 and ran for five times the time limit or $5T$ (950s). It took around 8 hours to run each instance for 31 times. As shown in Table 6.17, the algorithm is extendable as the best and average cost improved significantly when the runtime is extended. In fact, we managed to

obtain the best known results for the instances. It is important to note that we simply reset *runtime* in the algorithm without tuning any parameters, as is often required in a conventional SA e.g. decay rate. The p values ($0.000 < 0.05$) of t-tests reject the null hypotheses $H_0 : \mu_T = \mu_{5T}$ and revealed a statistically difference between the mean between the runtime of T and $5T$.

	runtime=T		*runtime*=$5T$		t-test
Instance	best	mean	best	mean	(p value)
Socha-L	165	206.61	103	**139.39**	0.000
ITC02-1	23	32.61	10	**21.03**	0.000
ITC07-1	0	307.55	0	**134.94**	0.000
Avg.	-	182.26	-	**98.45**	

Table 6.17: Comparison of soft constraint violations between SAR with a runtime of T and $5T$. n=31 runs.

6.2 Discussion

TSSP assisted SAR in obtaining the good results. As only a fraction of time is used by TSSP to find feasible solutions, more time is allocated for SAR to improve the soft constraint violations.

The right neighborhood structure composition used in SAR contributed to the good results. For Socha and ITC02 instances, the search spaces are well connected by transfer and swap operators. Therefore, a Kempe operator is redundant for these instances. Furthermore, the Kempe operator is computationally more expensive, thus reducing the number of transitions attempted. Meanwhile, for ITC07 instances, the search space is poorly connected by transfer and swap operators. Thus, a higher composition of a Kempe operator is worthwhile for the instances as it increases the connectivity of search space. In fact, ITC07 instances are more constrained compared to the instances of Socha and ITC07 as there are two additional hard constraints for ITC07 instances (order of events and preset time slots).

The neighborhood examination scheme applied in SAR, played a key role in achieving the good results. Attempts were made to deterministically move each event to each time slot (1-45) in each Markov chain. The scheme allows neighbors to be examined thoroughly and systematically with less redundancy.

SAR is able to estimate whether the search is stuck in a local optima, thus allows reheating to be applied at the right time. In addition, incremental reheating provides exploration opportunities for the search to escape from local optima while ensuring that the current solution does not stray too much thus preserving the previous search effort.

To highlight the importance of reheating in SAR, we compare the algorithm with reheating disabled, basic reheating and incremental reheating. The behavior (variation in temperature, current cost and best cost) of the algorithm for different reheating settings on selected instances are illustrated in Figures 6.1 to 6.18. The current and best cost for the algorithm with reheating disabled, becomes idle early indicating the search is stuck. Meanwhile, the current cost for the algorithm with basic reheating is active throughout the search. In a comparison, a lower soft constraint violation (best) is achieved by the algorithm with basic reheating. The SAR algorithm with incremental reheating performs equally well if not better than its counterpart with basic reheating. Incremental reheating is effective, particularly when the current cost is low as shown in Figures 6.3, 6.9 and 6.12.

The rigorous setting of the initial and end temperature in conventional SA is bypassed in SAR. We set the decay rate β and the constant C as 0.9995 and 0.01 respectively. Nonetheless, the values work well for the all the instances considered in this work.

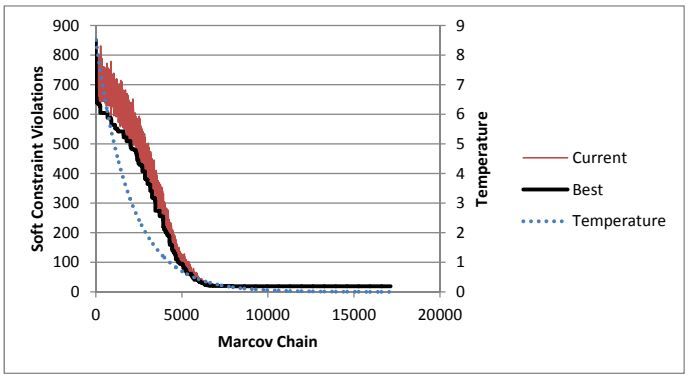

Figure 6.1: SAR (reheating disabled) on Socha-M1. Best=19.

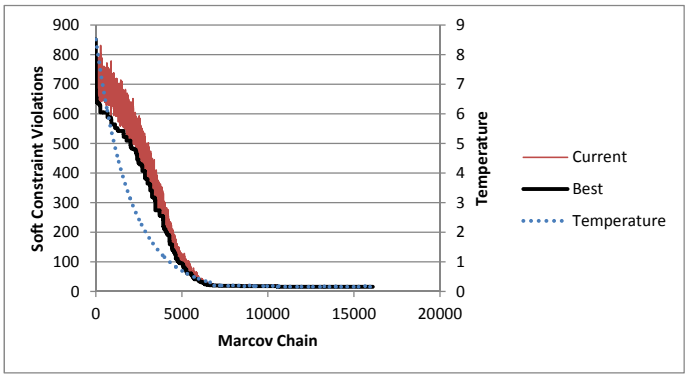

Figure 6.2: SAR (basic reheating) on Socha-M1. Best=16.

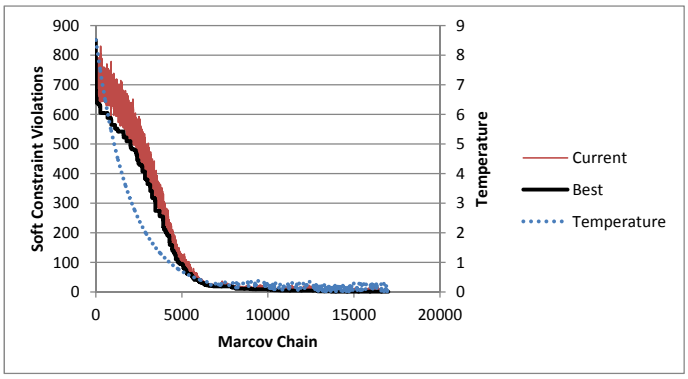

Figure 6.3: SAR (incremental reheating) on Socha-M1. Best=0.

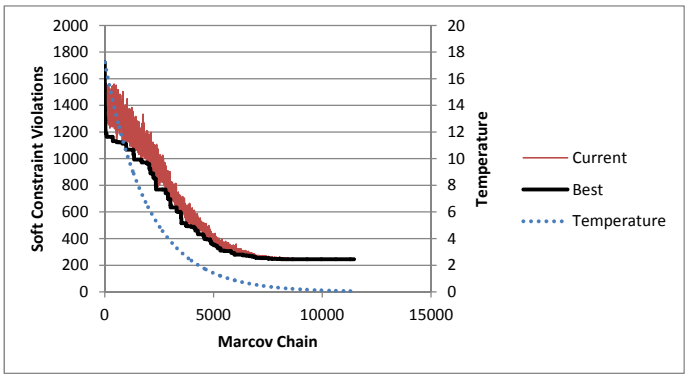

Figure 6.4: SAR (reheating disabled) on Socha-L. Best=245.

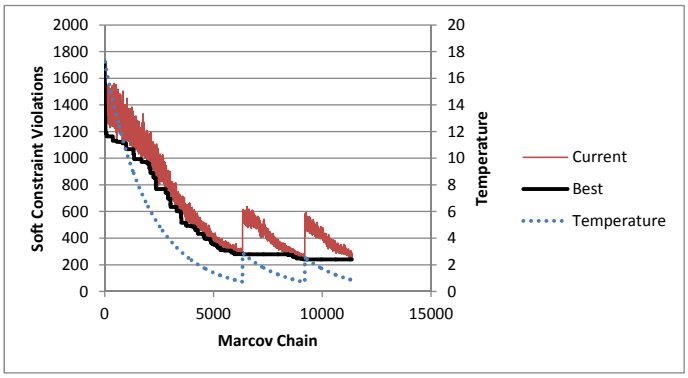

Figure 6.5: SAR (basic reheating) on Socha-L. Best=240.

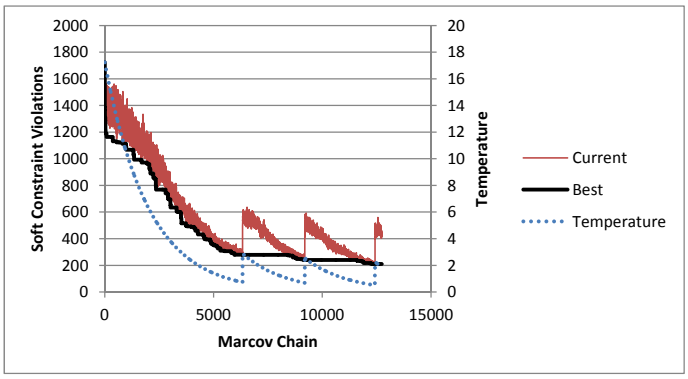

Figure 6.6: SAR (incremental reheating) on Socha-L. Best=210.

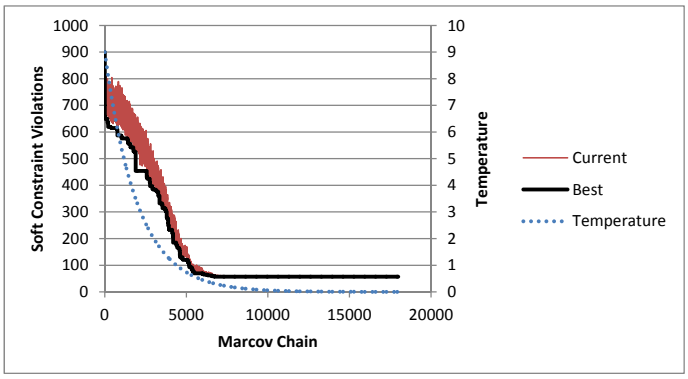

Figure 6.7: SAR (reheating disabled) on ITC02-1. Best=57.

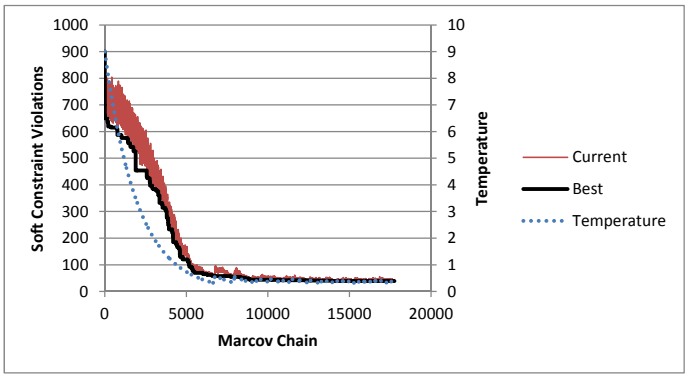

Figure 6.8: SAR (basic reheating) on ITC02-1. Best=39.

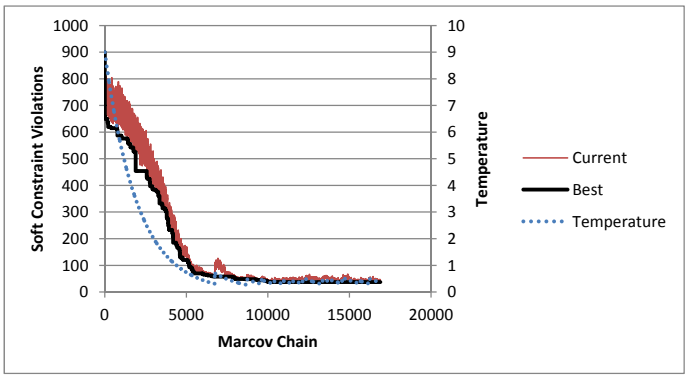

Figure 6.9: SAR (incremental reheating) on ITC02-1. Best=37.

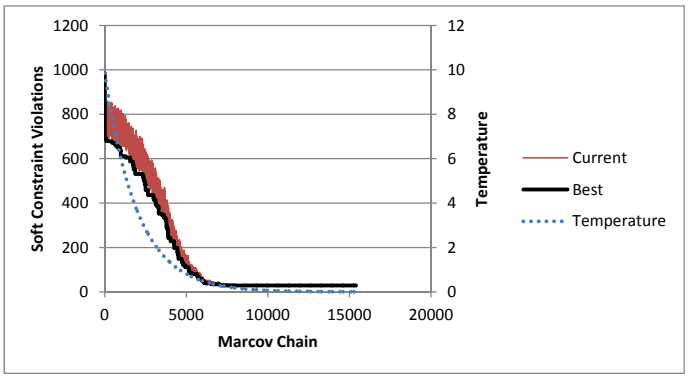

Figure 6.10: SAR (reheating disabled) on ITC02-11. Best=29.

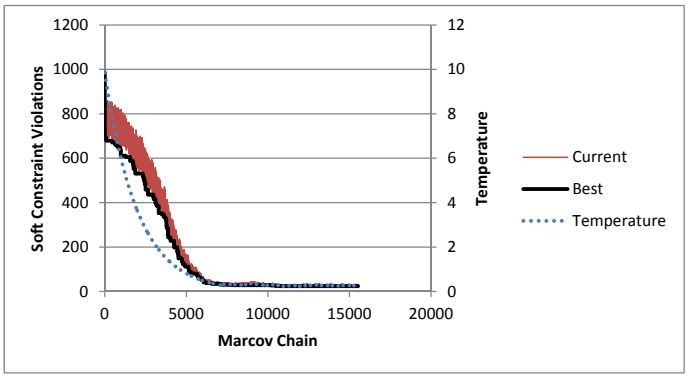

Figure 6.11: SAR (basic reheating) on ITC02-11. Best=24.

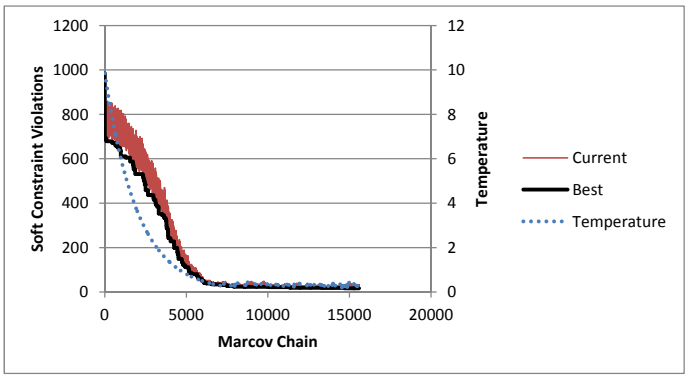

Figure 6.12: SAR (incremental reheating) on ITC02-11. Best=17.

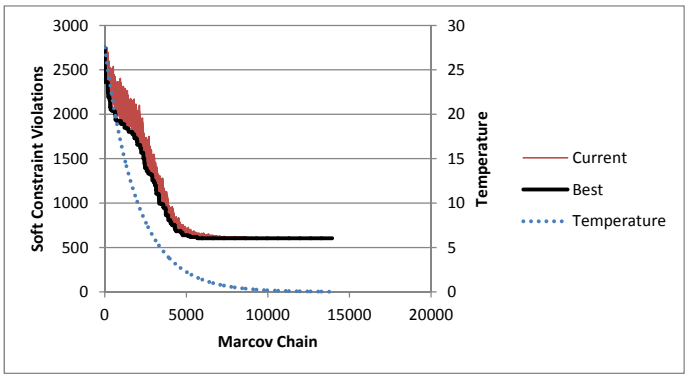

Figure 6.13: SAR (reheating disabled) on ITC07-1. Best=604.

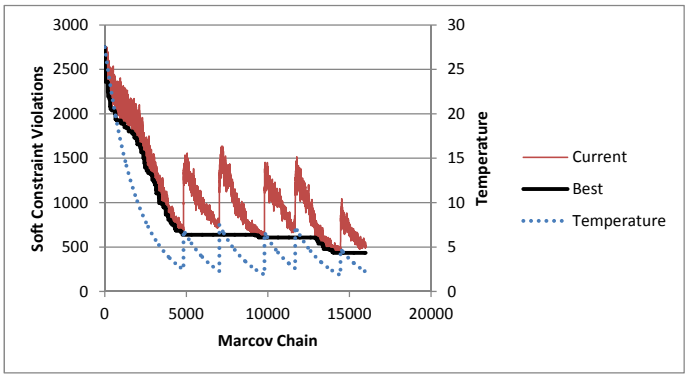

Figure 6.14: SAR (basic reheating) on ITC07-1. Best=435.

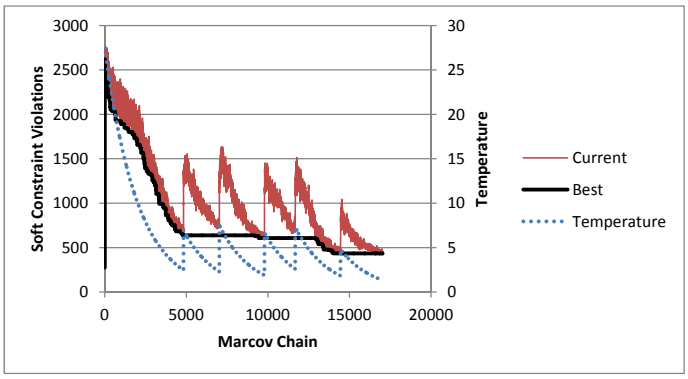

Figure 6.15: SAR (incremental reheating) on ITC07-1. Best=433.

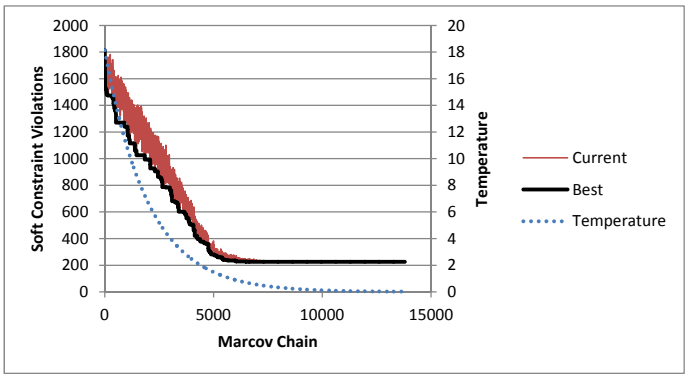

Figure 6.16: SAR (reheating disabled) on ITC07-13. Best=226.

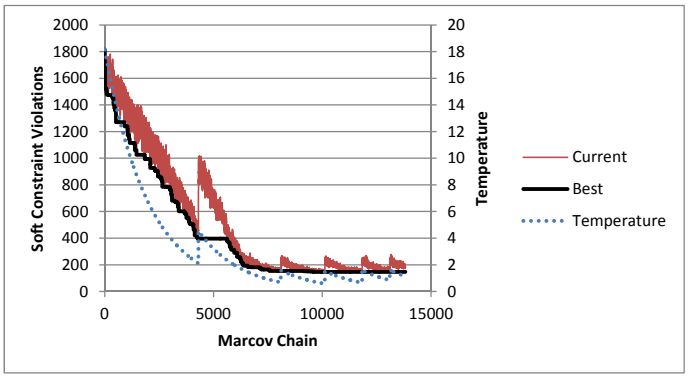

Figure 6.17: SAR (basic reheating) on ITC07-13. Best=147.

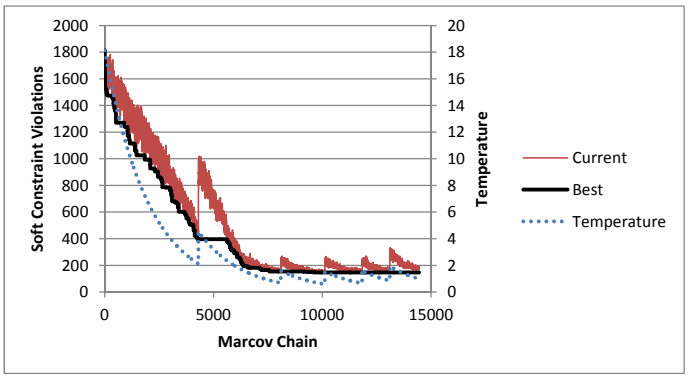

Figure 6.18: SAR (incremental reheating) on ITC07-13. Best=147.

6.3 Conclusion

We have compared the effectiveness of both individual and combination of neighborhood structures. The right combination of neighborhood structures is important in getting high quality solutions. It is more effective than any individual operator used alone. However, finding the right ratio for the operators is difficult as its effectiveness is dependent on the instances.

We analyzed various neighborhood examination schemes or the method of selecting events and time slots for move evaluation. We found that DE-DS (deterministic event and deterministic slot) is more effective than both RE-RS (random event and random slot) and DE-RS (deterministic event and random slot). DE-DS allows the search space to be thoroughly explored.

We presented the effect of basic reheating. The temperature is reheated to a value as a function of the current cost when the search is assumed stuck. In effect, exploration is guided by the current cost. The setup with reheating generated better results than its counterpart without reheating.

We tested three types of local optima detection. Type 2 (relative changes of current cost) seems to be more effective than Type 1 (absolute changes of current cost) and Type 3 (relative changes of current cost). A precise local optima estimator is vital to prevent unnecessary reheating which may affect the exploitation capability of the search.

We also compared the effect of incremental reheating where temperature is reheated to a higher value if the search is still stuck since the previous reheating. Slightly better results are observed when incremental reheating is used on the selected instances. Experimental results show that incremental reheating thrives while basic reheating suffers in terms of performance when the current cost is low. Incremental reheating helps the search to escape from local optima while ensuring that the current solution does not stray too much (preserving previous search effort).

Finally, we compared the performance of the SAR algorithm with state of the art methods. Competitive results in terms of soft constraint violations

are reported in all datasets tested. The behaviour of SAR and the effect of reheating are also displayed. Moreover, SAR is also shown to be extendable when the runtime is extended.

Chapter 7

Improving Quality: SAIRL Algorithm

In this chapter, we further enhance the SAR algorithm in terms of ease of use and performance (minimizing soft constraint violations). The work in this chapter has been submitted for peer review:

- Say Leng Goh, Graham Kendall, Nasser R. Sabar. Solving Post Enrolment Course Timetabling Problem using Simulated Annealing with Improved Reheating and Learning (SAIRL). Journal of Operational Research Society (JORS). Under review.

7.1 SA with Improved Reheating and Learning (SAIRL)

One drawback of SAR is having to preset the composition of neighborhood structures for the datasets in order to obtain good results. It is difficult to set the right composition as the effectiveness is dependent on the instance. Another drawback of SAR is the limitation of using the current cost exclusively to determine the level of reheated temperature as different instances may require different level of exploration to search effectively. We propose several enhancements based on these shortcomings. We term the improved algorithm Simulated Annealing with Improved Reheating and Learning (SAIRL).

7.1.1 Algorithm Description

The SAIRL algorithm is shown in Algorithm 39.

Algorithm 39

```
1: procedure SAIRL(current, E)
2:     temp ← f(current) × C
3:     heat ← 0
4:     best ← current
5:     previousCost ← f(current)
6:     currentStagnantCount ← 0
7:     stuckedBestCost ← f(current)
8:     stuckedCurrentCost ← f(current)
9:
10:    while current is not optimal AND time.elapsed() < runtime do
11:        for all e ∈ E do
12:            moved ← false
13:            for slot = 1 to 45 do
14:                n ← selectNeighbourhoodStructure()
15:                visit_n ++
16:                candidate ← getCandidate(current, e, slot, n)
17:                if candidate exists then
18:                    if random[0,1) ≤ exp(−(f(candidate)−f(current))/temp) then
19:                        moved ← true
20:                        current ← candidate
21:                        if f(current) < f(best) then
22:                            best ← current
23:                        end if
24:                        update value_n
25:                    else
26:                        update value_n
27:                    end if
28:                else
29:                    update value_n
30:                end if
31:                if moved then
32:                    break
33:                end if
34:            end for
35:        end for
```

```
36:         if STUCK(f(current), previousCost, currentStagnantCount) then
37:             if f(best) = stuckBestCost then
38:                 if f(current) − stuckCurrentCost < 2% then
39:                     heat = heat + 1
40:                 else
41:                     heat ← 0
42:                 end if
43:             else
44:                 heat ← 0
45:             end if
46:             temp ← [heat × 0.2 × f(current) + f(current)] × Δf̄ × D
47:             stuckBestCost ← f(best)
48:             stuckCurrentCost ← f(current)
49:         else
50:             temp ← temp × β
51:         end if
52:         previousCost ← f(current)
53:     end while
54: end procedure
```

We propose a method based on reinforcement learning (RL) to obtain a balanced composition of the neighborhood structures. The method is inspired by the observation that different neighbourhood structures have different acceptance ratio and computational cost for different instances. Some neighbourhood structures may have lower acceptance ratio but are less computationally expensive which allows more transitions per time unit. Therefore, the objective is to maximize the number of accepted moves per time unit.

In our implementation, a $visit_n$ and a $value_n$ are maintained for each neighborhood structure n. $visit_n$ is incremented by 1 each time the neighborhood structure n is selected (line 15). Meanwhile, $value_n$ is updated (lines 24, 26 and 29) as a cumulative mean of rewards:

$$value_n \leftarrow value_n + \frac{reward - value_n}{visit_n} \tag{7.1}$$

The reward is defined as:

$$reward = \begin{cases} 0, & \text{if candidate is accepted} \\ \text{CPU time}, & \text{otherwise} \end{cases} \tag{7.2}$$

A reward of 0 is awarded to the neighborhood structure n if the candidate solution is accepted (line 24). Otherwise, the neighborhood structure n is penalized with CPU time (elapsed time since selection) if the candidate is rejected (line 26) or the candidate does not exist because a move is not feasible (line 29).

Initially, all neighborhood structures have an equal probability of being selected. Over time, the probability varies according to:

$$P_n = \frac{\frac{1}{value_n}}{\sum_{n=1}^{N} \frac{1}{value_n}} \tag{7.3}$$

Roulette Wheel (RW) based selection is used to ensure the less favorable neighbourhood structures can still be selected but the better neighbourhood structures are more likely to be selected. Fig. 7.1 shows the interaction between agent and environment in our application of RL.

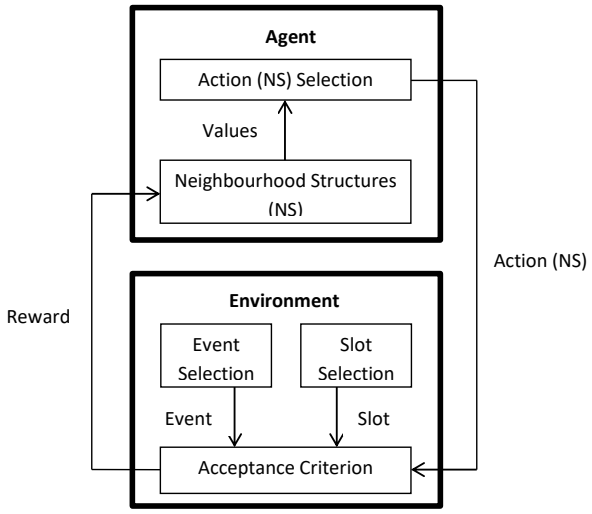

Figure 7.1: The interaction between agent and environment in RL.

Another enhancement that we propose to SAR is for the temperature reheating function. As the acceptance probability of an uphill move is determined by the cost changes of the move and temperature, intuitively it is a good idea to

incorporate the average cost changes of uphill moves ($\overline{\Delta f}$) into the temperature reheating function:

$$temp \leftarrow [heat \times 0.2 \times f(current) + f(current)] \times \overline{\Delta f} \times D \tag{7.4}$$

D is a constant which determines the exploration level of the reheated temperature. We set the decay rate β to 0.9995 and the constant D to 0.001. To allow a fair comparison between SAR and SAIRL, the initial temperature is set to the same value used in SAR where the initial cost is multiplied by the constant C=0.01 (1% of the initial cost). The same settings are used across all instances in our experiments.

7.1.2 Experimental Results

7.1.2.1 The Effect of Learning

In this section, we compare the results of SAR and SARL for selected instances. For SAR, neighborhood structure composition is set manually for specific datasets. The composition are 70:29:1 (Socha/ITC02 instances) and 70:20:10 (ITC07 instances) for Tansfer:Swap:Kempe operators. Meanwhile, a reinforcement learning based method is used in SARL to optimize the composition as the search progresses. SARL not only eliminates the requirement for manual setting of neighborhood structure composition but also improves the average of means for the selected instances as evident in Table 7.1.

	SAR		SARL	
Instance	best	mean	best	mean
Socha-L	165	206.61	151	**204.16**
ITC02-1	23	**32.61**	28	36.45
ITC07-1	0	307.55	0	**277.45**
Avg.	-	182.26	-	**172.69**

Table 7.1: Comparison of soft constraint violations between SAR and SARL on selected instances. n=31 runs.

7.1.2.2 The Effect of Improved Reheating

We also investigate the effect of incorporating the average cost changes into the reheated temperature function. The average cost change varies for each instance as shown in Figures 7.2, 7.3 and 7.4. It provides an insight to the gradient of the search landscape. SAIRL further improved the average of means for selected

instances as shown in Table 7.2. A notable improvement can be seen for instance ITC07-01.

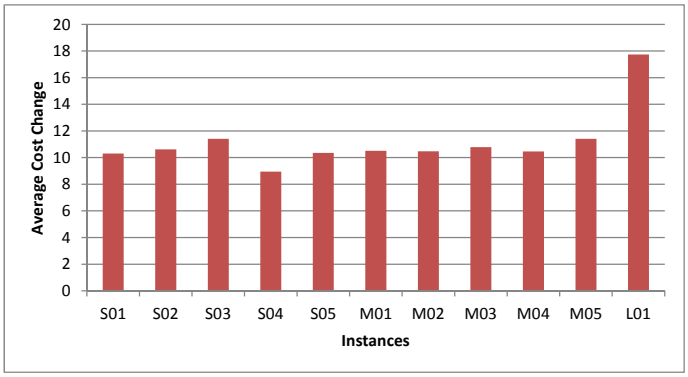

Figure 7.2: Average Cost Changes for Socha instances

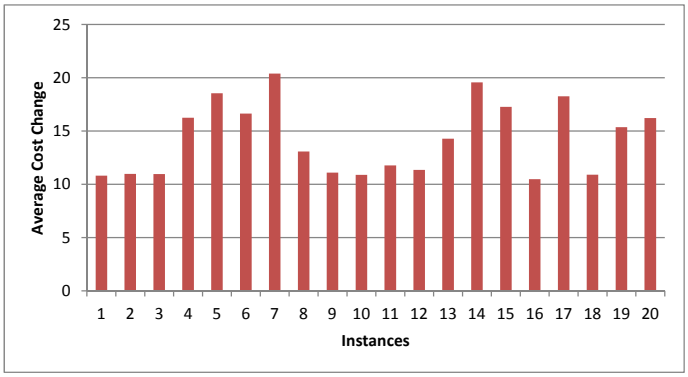

Figure 7.3: Average Cost Changes for ITC02 instances

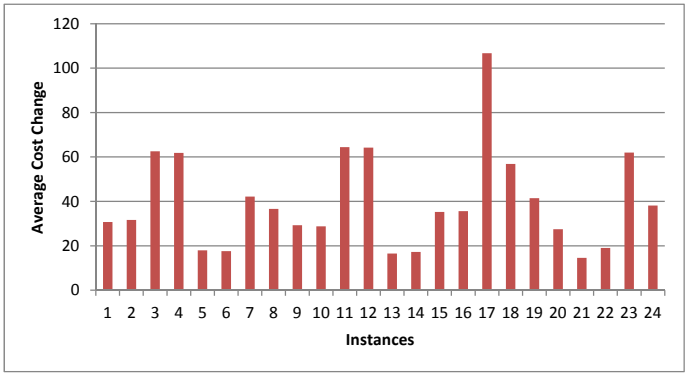

Figure 7.4: Average Cost Changes for ITC07 instances

	SARL		SAIRL	
Instance	best	mean	best	mean
Socha-L	151	**204.16**	181	215.19
ITC02-1	28	36.45	26	**35.71**
ITC07-1	0	277.45	0	**209.39**
Avg.	-	172.69	-	**153.87**

Table 7.2: Comparison of soft constraint violations between SARL and SAIRL on selected instances. n=31 runs.

We tested several values for D as shown in Table 7.3. The values of 0.0005, 0.001 and 0.002 recorded the lowest means for the instances Socha-L, ITC02-1 and ITC07-1 respectively. The value of 0.001 is used onwards as it has the lowest average of means.

	Values of D					
	0.0005		0.001		0.002	
Instance	best	mean	best	mean	best	mean
Socha-L	150	**205.42**	181	215.19	174	234.35
ITC02-1	28	39.94	26	**35.71**	36	48.97
ITC07-1	0	327.77	0	209.39	0	**188.00**
Avg.	-	191.04	-	**153.87**	0	157.11

Table 7.3: The effect of basic reheating upon soft constraint violations on selected instances. n=31 runs.

7.1.2.3 Comparing SAIRL with SAR

We compare the performance of SAR and SAIRL in minimizing the soft constraint violations. For Socha instances, SAIRL is comparable to SAR as shown in Table 7.4. The p values reveal that there is no significant difference between the means of SAR and SAIRL for all the instances except M2 where SAR is better than SAIRL.

Instance	SAR	SAIRL	t-test (p value)
S1	**0**(**0.0**)	**0**(**0.0**)	-
S2	**0**(**0.0**)	**0**(**0.0**)	-
S3	**0**(**0.0**)	**0**(**0.0**)	-
S4	**0**(**0.0**)	**0**(**0.0**)	-
S5	**0**(**0.0**)	**0**(**0.0**)	-
M1	**0**(**1.5**)	**0**(2.32)	0.057
M2	**0**(**2.2**)	**0**(3.58)	**0.007**
M3	7(**13.4**)	**6**(14.39)	0.443
M4	**0**(**0.7**)	**0**(1.35)	0.073
M5	**0**(**1.2**)	**0**(1.42)	0.600
L	**165**(**206.6**)	181(215.19)	0.127

Table 7.4: Comparison between SAR and SAIRL on Socha instances. Depicted is best(mean) of soft constraint violations. n=31 runs.

Results comparison between SAIRL and SAR for ITC02 instances is shown in Table 7.5. The t-tests show that SAR performed better than SAIRL for instances 1, 2, 9, 16, 18. Meanwhile, SAIRL is more effective for instances 5 and 17. There is no significant difference between the means for the rest of the instances.

Instance	SAR	SAIRL	t-test (p value)
1	**23**(**32.6**)	26(35.7)	**0.040**
2	7(**13.7**)	**6**(16.3)	**0.031**
3	**26**(**36.4**)	27(38.2)	0.291
4	50(**63.1**)	**47**(69.0)	0.062
5	38(58.6)	**36**(**51.8**)	**0.005**
6	**0**(**0.8**)	**0**(**0.8**)	0.826
7	**0**(2.6)	**0**(**2.4**)	0.579
8	**0**(**1.4**)	**0**(1.5)	0.782
9	**0**(**4.6**)	**0**(6.4)	**0.025**
10	28(40.9)	**22**(**40.4**)	0.761
11	**10**(**17.7**)	**10**(19.0)	0.318
12	53(64.5)	**47**(**64.1**)	0.881
13	38(53.3)	**33**(**51.0**)	0.297
14	5(**12.9**)	**4**(13.6)	0.587
15	**0**(**4.0**)	**0**(4.8)	0.234
16	**0**(**0.5**)	**0**(2.2)	**0.000**
17	26(41.6)	**25**(**36.8**)	**0.044**
18	**2**(**9.7**)	3(12.5)	**0.005**
19	**11**(**24.7**)	15(25.6)	0.577
20	**0**(**0.0**)	**0**(**0.0**)	-

Table 7.5: Comparison between SAR and SAIRL on ITC02 instances. Depicted is best(mean) of soft constraint violations. n=31 runs.

For ITC07 instances, SAIRL performed significantly better compared to SAR for instances 1, 2, 3, 9, 11, 15, 16, 19, 24 as shown in Table 7.6. SAR is better than SAIRL for instances 14 and 23. No significant difference is evident between the means for the rest of the instances.

Instance	SAR	SAIRL	t-test (p value)
1	**0**(307.6)	**0**(**209.4**)	**0.025**
2	**0**(63.4)	**0**(**10.1**)	**0.048**
3	163(199.4)	**141**(**188.6**)	**0.050**
4	242(328.8)	**192**(**320.9**)	0.456
5	**0**(**2.7**)	**0**(2.9)	0.845
6	**0**(**33.2**)	**0**(54.7)	0.074
7	5(18.0)	**4**(**14.5**)	0.614
8	**0**(**0.0**)	**0**(1.6)	0.156
9	**0**(100.7)	**0**(**15.2**)	**0.009**
10	**0**(65.3)	**0**(**30.5**)	0.160
11	161(244.3)	**136**(**201.6**)	**0.001**
12	**0**(318.2)	**0**(**303.5**)	0.641
13	**0**(99.5)	**0**(**90.4**)	0.605
14	**0**(**0.2**)	**0**(25.6)	**0.001**
15	**0**(192.0)	**0**(**12.5**)	**0.000**
16	10(105.8)	**0**(**45.8**)	**0.000**
17	**0**(0.8)	**0**(**0.5**)	0.590
18	**0**(12.5)	**0**(**7.7**)	0.366
19	**0**(516.7)	**0**(**11.0**)	**0.000**
20	586(**650.7**)	**555**(664.0)	0.280
21	**0**(**12.5**)	**0**(25.7)	0.071
22	1(136.0)	**0**(**5.8**)	0.099
23	**11**(**504.4**)	56(713.6)	**0.005**
24	5(192.6)	**0**(**77.5**)	**0.000**

Table 7.6: Comparison between SAR and SAIRL on ITC07 instances. Depicted is best(mean) of soft constraint violations. n=31 runs.

7.1.2.4 Comparing SAIRL with State of the Art Methods

We now compare SAIRL with the best results in the literature. Table 7.7 summarizes the details of the solvers we use for comparison.

Solver	Technique	Reference
A	Ant System	Socha et al. [164]
B	Tabu Search Hyperheuristic	Burke et al. [35]
C	Extended Great Deluge	McMullan [129]
D	Great Deluge + Tabu Search	Abdullah et al. [7]
E	Non Linear Great Deluge + Learning	Obit et al. [136]
F	Fish Swarm	Turabieh et al. [176]
G	Round Robin Multi Algorithms	Shaker and Abdullah [161]
H	Honey Bee Mating	Sabar et al. [155]
I	Simulated Annealing	Ceschia et al. [48]
J1	Simulated Annealing	Kostuch [109]
J2	Simulated Annealing	Kostuch [110]
K	Tabu Search	Cordeau et al. [55]
L	Great Deluge	Burke et al. [27]
M	Local Search + Tabu Search	DiGaspero and Schaerf [63]
N	Hybrid Algorithm	Chiarandini et al. [52]
O	Simulated Annealing	Cambazard et al. [41]
P	Ant Colony Optimization	Nothegger et al. [135]
Q	Simulated Annealing	Lewis and Thompson [118]
R	Simulated Annealing with Reheating (SAR)	Goh et al. [88]

Table 7.7: Solver details

SAIRL outperformed all the other solvers for all Socha instances except solver R which we attempt to improve in this work as shown in Table 7.8. Both SAIRL and solver R found optimal solutions for 9 out of 11 instances. In addition, SAIRL achieved a new best result for instance M3.

Results comparison for ITC02 is given in Table 7.9. Our results are competitive or better than the other solvers on all the instances. In fact, SAIRL managed to get optimal solutions for 7 out of 20 instances in comparison to solver J2 (four) and solver R (seven). Furthermore, SAIRL obtained four new best results (instance 5, 12, 14 and 17) and four best means (instance 5, 7, 12 and 17).

Table 7.10 shows the results comparison for ITC07. Our results are competitive compared to the other solvers. SAIRL found eighteen optimal solutions compared to solver Q (seventeen) and solver R (fifteen). SAIRL achieved one new best result (instance 22) and ten best means (instance 1, 2, 9, 11, 12, 15, 16, 19, 22, 24). Solver P was not executed on instances 17-24.

	Solver										
Instance	A	B	C	D	E	F	G	H	I	R	SAIRL
S1	1	1	**0**(0.8)	**0**	**0**	**0**	**0**	**0**	**0**(**0.0**)	**0**(**0.0**)	**0**(**0.0**)
S2	3	2	**0**(2.0)	**0**	**0**	**0**	**0**	**0**	**0**(**0.0**)	**0**(**0.0**)	**0**(**0.0**)
S3	1	**0**	**0**(1.3)	**0**	**0**	**0**	**0**	**0**	**0**(**0.0**)	**0**(**0.0**)	**0**(**0.0**)
S4	1	1	**0**(1.0)	**0**	**0**	**0**	**0**	**0**	**0**(0.1)	**0**(**0.0**)	**0**(**0.0**)
S5	**0**	**0**	**0**(0.2)	**0**	**0**	**0**	**0**	**0**	**0**(**0.0**)	**0**(**0.0**)	**0**(**0.0**)
M1	195	146	80(101.4)	78	38	45	117	75	9(26.5)	**0**(**1.5**)	**0**(2.32)
M2	184	173	105(116.9)	92	37	40	108	88	15(25.9)	**0**(**2.2**)	**0**(3.58)
M3	248	267	139(162.1)	135	60	61	135	129	36(49.0)	7(**13.4**)	**6**(14.39)
M4	164.5	169	88(108.8)	75	39	35	75	74	12(23.8)	**0**(**0.7**)	**0**(1.35)
M5	219.5	303	88(119.7)	68	55	49	160	64	3(10.9)	**0**(**1.2**)	**0**(1.42)
L	851.1	1166	730(834.1)	556	638	407	589	523	208(259.8)	**165**(**206.6**)	181(215.19)

Table 7.8: Comparing SAIRL with other solvers on Socha instances. Depicted is best(mean) of soft constraint violations. n=31 runs. Note that some authors only reported their best results.

Instance	Solver								
	J1	K	L	M	N	J2	I	R	SAIRL
1	45	61	85	63	45	**16**(**30.2**)	45(57.1)	23(32.6)	26(35.7)
2	25	39	42	46	14	**2**(**11.4**)	20(33.2)	7(13.7)	6(16.3)
3	65	77	84	96	45	**17**(**31.0**)	43(53.2)	26(36.4)	27(38.2)
4	115	160	119	166	71	**34**(**60.8**)	87(109.9)	50(63.1)	47(69.0)
5	102	161	77	203	59	42(72.1)	71(91.7)	38(58.6)	**36**(**51.8**)
6	13	42	6	92	1	**0**(2.4)	2(14.1)	**0**(**0.8**)	**0**(**0.8**)
7	44	52	12	118	3	2(8.9)	2(13.7)	**0**(2.6)	**0**(**2.4**)
8	29	54	32	66	1	**0**(2.0)	9(20.0)	**0**(**1.4**)	**0**(1.5)
9	17	50	184	51	8	1(5.8)	15(21.9)	**0**(**4.6**)	**0**(6.4)
10	61	72	90	81	52	**21**(**35.0**)	41(60.7)	28(40.9)	22(40.4)
11	44	53	73	65	30	**5**(**12.9**)	24(38.2)	10(17.7)	10(19.0)
12	107	110	79	119	75	55(76.3)	62(83.7)	53(64.5)	**47**(**64.1**)
13	78	109	91	160	55	**31**(**47.1**)	59(78.0)	38(53.3)	33(51.0)
14	52	93	36	197	18	11(22.3)	21(34.2)	5(**12.9**)	**4**(13.6)
15	24	62	27	114	8	2(8.4)	6(11.8)	**0**(**4.0**)	**0**(4.8)
16	22	34	300	38	55	**0**(3.4)	6(16.7)	**0**(**0.5**)	**0**(2.2)
17	86	114	79	212	46	37(54.0)	42(56.5)	26(41.6)	**25**(**36.8**)
18	31	38	39	40	24	4(**9.4**)	11(25.9)	**2**(9.7)	3(12.5)
19	44	128	86	185	33	**7**(**16.4**)	56(73.0)	11(24.7)	15(25.6)
20	7	26	**0**	17	**0**	**0**(0.5)	**0**(1.8)	**0**(**0.0**)	**0**(**0.0**)

Table 7.9: Comparing SAIRL with other solvers on ITC02 instances. Depicted is best(mean) of soft constraint violations. n=31 runs. Note that some authors only reported their best results.

Instance	Solver				
	P	I	Q	R	SAIRL
1	**0**(613.0)	59(399.2)	**0**(377.0)	**0**(307.6)	**0**(**209.4**)
2	**0**(556.0)	**0**(142.2)	**0**(382.2)	**0**(63.4)	**0**(**10.1**)
3	**110**(680.0)	148(209.9)	122(**181.8**)	163(199.4)	141(188.6)
4	53(580.0)	25(349.6)	**18**(**319.4**)	242(328.8)	192(320.9)
5	13(92.0)	**0**(7.7)	**0**(7.5)	**0**(**2.7**)	**0**(2.9)
6	**0**(212.0)	**0**(**8.6**)	**0**(22.8)	**0**(33.2)	**0**(54.7)
7	**0**(4.0)	**0**(4.9)	**0**(5.5)	5(18.0)	4(14.5)
8	**0**(61.0)	**0**(1.5)	**0**(0.6)	**0**(**0.0**)	**0**(1.6)
9	**0**(202.0)	**0**(258.8)	**0**(514.4)	**0**(100.7)	**0**(**15.2**)
10	**0**(**4.0**)	3(186.4)	**0**(1202.4)	**0**(65.3)	**0**(30.5)
11	143(774.0)	142(269.5)	**48**(202.6)	161(244.3)	136(**201.6**)
12	**0**(538.0)	267(400.0)	**0**(340.2)	**0**(318.2)	**0**(**303.5**)
13	5(360.0)	1(120.0)	**0**(79.0)	**0**(99.5)	**0**(90.4)
14	**0**(41.0)	**0**(3.6)	**0**(0.5)	**0**(**0.2**)	**0**(25.6)
15	**0**(29.0)	**0**(48.0)	**0**(139.9)	**0**(192.0)	**0**(**12.5**)
16	**0**(101.0)	**0**(50.1)	**0**(105.2)	10(105.8)	**0**(**45.8**)
17	-	**0**(**0**)	**0**(0.1)	**0**(0.8)	**0**(0.5)
18	-	**0**(41.1)	**0**(**2.2**)	**0**(12.5)	**0**(7.7)
19	-	**0**(951.5)	**0**(346.1)	**0**(516.7)	**0**(**11.0**)
20	-	543(700.2)	557(724.5)	586(**650.7**)	555(664.0)
21	-	5(35.9)	1(32.1)	**0**(**12.5**)	**0**(25.7)
22	-	5(19.9)	4(1790.1)	1(136.0)	**0**(**5.8**)
23	-	1292(1707.7)	**0**(514.1)	11(**504.4**)	56(713.6)
24	-	**0**(105.3)	18(328.2)	5(192.6)	**0**(**77.5**)

Table 7.10: Comparing SAIRL with other solvers on ITC07 instances. Depicted is best(mean) of soft constraint violations. n=31 runs.

7.1.2.5 Extended Runtime for SAIRL

Lastly, we performed some experiments to see the effects of an extended runtime with regard to soft constraint violations on selected instances. The algorithm was ran for 31 times with a time limit of $5\,T$ (950s). As evident in Table 7.11, the algorithm is extendable as the best and average cost improved significantly when the runtime is extended. Note that *runtime* is simply reset without tuning any parameters. The p values ($0.000 < 0.05$) of t-tests reject the null hypotheses $H_0 : \mu_T = \mu_{5T}$ and revealed a statistical difference between the mean between the runtime of T and $5T$. The soft constraint violations for SAIRL with extended runtime on Socha-L, ITC02-1 and ITC07-1 instances are illustrated in Figures 7.5, 7.6 and 7.7. The respective descriptive statistics are given in Tables 7.12, 7.13 and 7.14. Note that the circles and the stars in the box plots are mild and

extreme outliers.

	runtime=T		*runtime*=$5T$		t-test
Instance	best	mean	best	mean	(p value)
Socha-L	181	215.19	157	**190.42**	0.000
ITC02-1	26	35.71	11	**20.84**	0.000
ITC07-1	0	209.39	0	**23.06**	0.000
Avg.	-	153.87	-	**78.11**	

Table 7.11: Comparison of soft constraint violations between SAIRL with a runtime of T and $5T$ on selected instances. n=31 runs.

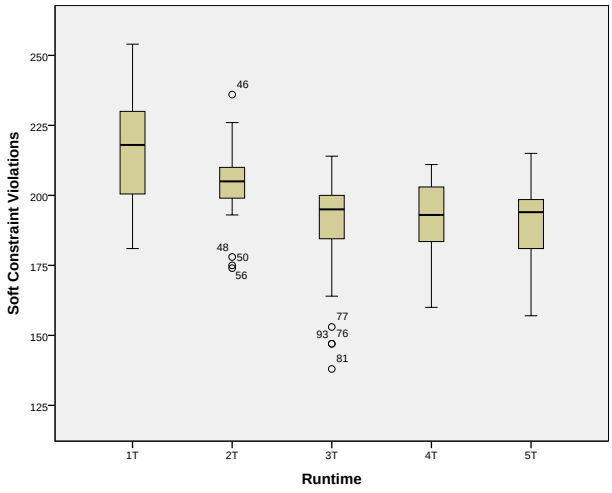

Figure 7.5: Box plot showing the soft constraint violations for SAIRL with an extended runtime on Socha-L instance. n=31 runs.

	Runtime				
Unassigned	1T	2T	3T	4T	5T
Min	181	174	138	160	157
Max	254	236	214	211	215
Median	218.00	205.00	195.00	193.00	194.00
Mean	215.19	204.97	189.65	191.58	190.42

Table 7.12: Descriptive statistics for SAIRL with an extended runtime on Socha-L instance. n=31 runs.

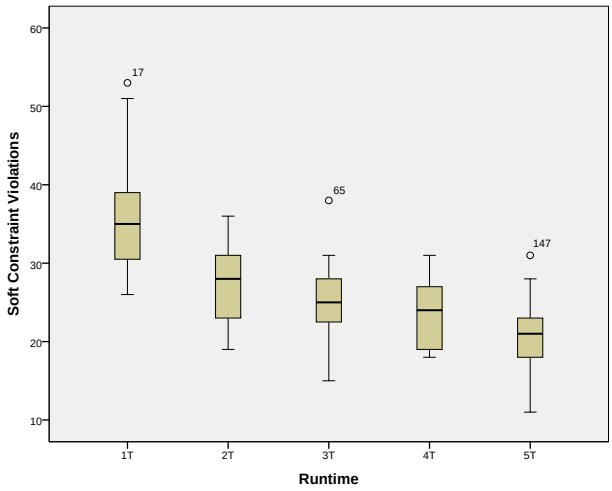

Figure 7.6: Box plot showing the soft constraint violations for SAIRL with an extended runtime on ITC02-1 instance. n=31 runs.

	Runtime				
Unassigned	1T	2T	3T	4T	5T
Min	26	19	15	18	11
Max	53	36	38	31	31
Median	35.00	28.00	25.00	24.00	21.00
Mean	35.71	27.84	25.03	23.65	20.84

Table 7.13: Descriptive statistics for SAIRL with an extended runtime on ITC02-1 instance. n=31 runs.

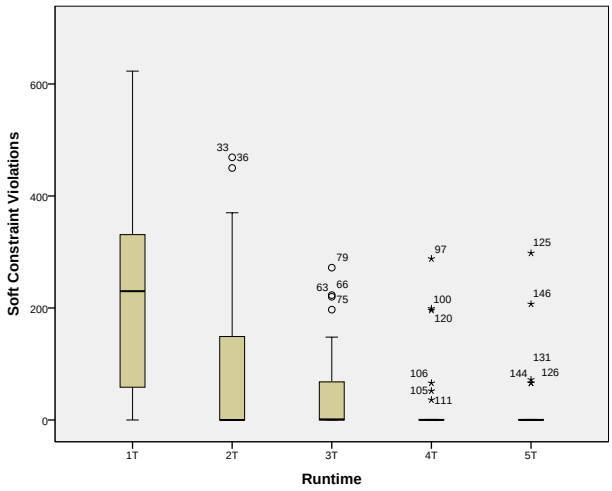

Figure 7.7: Box plot showing the soft constraint violations for SAIRL with an extended runtime on ITC07-1 instance. n=31 runs.

	Runtime				
Unassigned	1T	2T	3T	4T	5T
Min	0	0	0	0	0
Max	623	469	272	288	298
Median	230.00	0.00	1.00	0.00	0.00
Mean	209.39	90.23	48.29	27.23	23.06

Table 7.14: Descriptive statistics for SAIRL with an extended runtime on ITC07-1 instance. n=31 runs.

We also compare the results of extended runtime (five times the time limit or $5\,T$) between SAR and SAIRL for selected instances. As it took around 8 hours to run each instance for 31 times, therefore, we selected only one instance from each dataset namely Socha-L, ITC02-1 and ITC07-1. SAIRL is comparable to SAR as shown is Table 7.15. In fact, SAIRL is preferred based on the average of means for the selected instances. There is no significant difference between the means of both algorithms for ITC02-1. SAIRL performed better than SAR for instance ICT2007-1. Meanwhile, SAR is more effective than SAIRL for instance Socha-L. One possible explanation is, average cost changes can be distorted by very large or very small cost changes. Therefore, the information on the gradient of the search landscape can be distorted as well as the reheated temperature

since the average of cost changes is incorporated into the reheated temperature function.

	SAR ($5\,T$)		SAIRL ($5\,T$)		t-test
Instance	best	mean	best	mean	(p value)
Socha-L	103	**139.39**	157	190.42	0.000
ITC02-1	10	21.03	11	**20.84**	0.867
ITC07-1	0	134.94	0	**23.06**	0.000
Avg.	-	98.45	-	**78.11**	

Table 7.15: Comparison of soft constraint violations between SAR ($5\,T$) and SAIRL ($5\,T$) on selected instances. n=31 runs.

7.2 Discussion

In order for a conventional SA to produce good results, certain parameters have to be tuned, even for specific instances e.g. initial temperature, end temperature, Marcov chain length and decay rate. SAIRL, proposed in this chapter, not only eliminates the requirement for tuning those parameters but also eliminates the selection of neighborhood structure composition (dataset specific) in SAR. Nevertheless, the algorithm produces superior results compared to the state of the art methods which were either heavily tuned or limited in terms of datasets considered. For SAIRL, we only have to set the decay rate β to 0.9995 and the constant D to 0.001. The same settings were used, and worked well, for all the instances without tuning. Meanwhile the initial temperature is simply set as 1% of the initial cost. Unlike conventional SA, the initial temperature is not critical for SAIRL as reheating allows search exploration to reset when stuck.

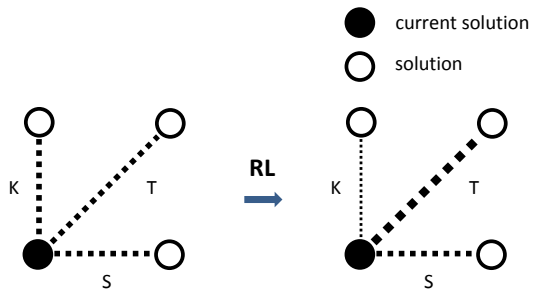

Figure 7.8: The use of Reinforcement Learning (RL) to adjust the neighborhood structure composition.

Multiple operators are often better than any single operator being used alone because the solution space is more connected when the operators are assumed to have equal acceptance ratio and computational cost. However, selecting operators with equal probability is suboptimal because in reality, the operators have different acceptance ratio and computational cost for different data instances. The effect of RL is shown in Figure 7.8. We use dashed instead of solid lines to indicate that the solution space is not necessarily connected by any of the operators as move acceptance is determined by feasibility as well as the temperature. Meanwhile, the thickness of the dashed lines represent the selection probabilities for the operators. Initially, the operators have an equal chance of being selected. As the search progresses, the probabilities are increased or decreased depending on the acceptance ratio and computational cost of the operators. Unlike other methods, which reward operators that improve the current or best solution, our RL based methodology rewards operators that change the current solution (improving moves or equal cost moves or worsening moves). Effectively, operators with relatively high value (cumulative mean of rewards) will have a higher tendency of being selected. The RW selection that we use prevents the domination of any operator as low valued operators can still be selected. As a result, the number of transitions (accepted moves) per time unit is maximized. The connectivity of the solution space is improved. The movement of neighborhood structure composition for Socha-L, ITC02-1 and ITC07-1 is shown in Fig. 7.9 , 7.10 and 7.11 respectively. Kempe operator seems more worthwhile for ITC07-1 compared to Socha-L and ITC02-1.

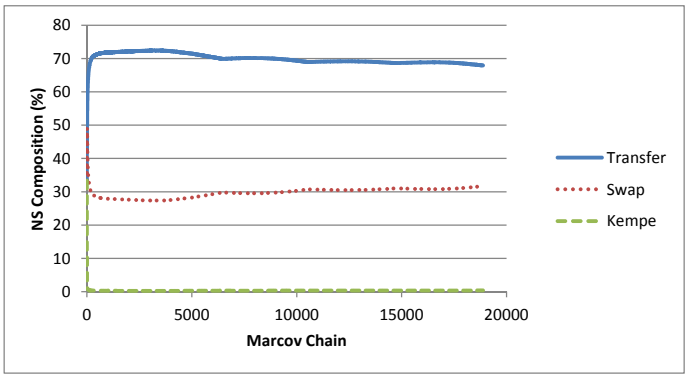

Figure 7.9: Movement of neighborhood structure composition for Socha-L

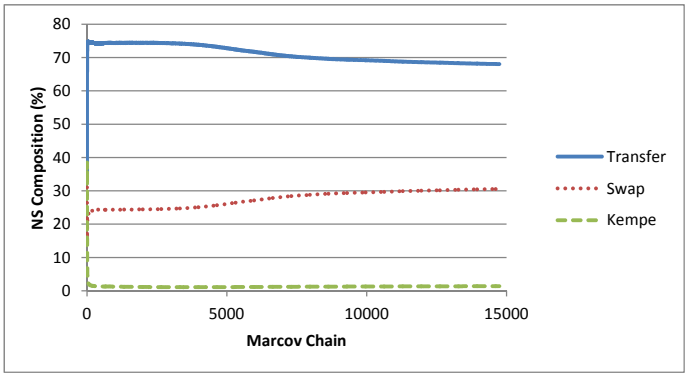

Figure 7.10: Movement of neighborhood structure composition for ITC02-1

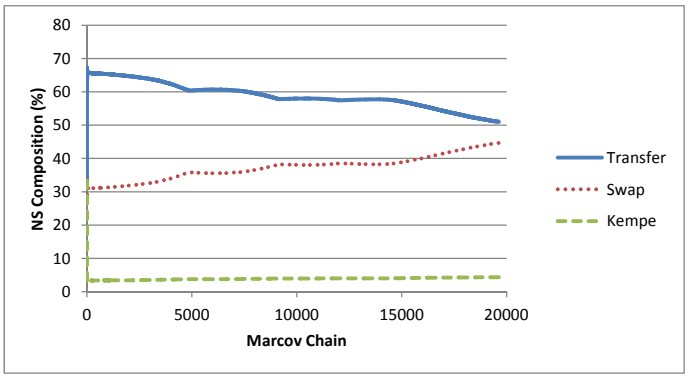

Figure 7.11: Movement of neighborhood structure composition for ITC07-1

In SAR, the reheated temperature is set proportional to the current cost. When the current cost is low, the temperature is set proportionally low. In effect, the search is guided to operate in the vicinity of the current solution in the hope of finding the optimal solution (exploitation). Meanwhile, when the current cost is high, the temperature is set proportionally high and the search is allowed to explore more. However, setting the reheated temperature based on the current cost alone is not sufficient as the search landscape for each instance is different. In SAIRL, the average cost change (which provides insight into the gradient of the search landscape) is incorporated into the reheated temperature function since the temperature determines the acceptance of uphill moves based on the cost changes. In effect, we are using the information on the gradient of the search landscape to determine the exploration level for the search.

7.3 Conclusion

We have compared the results of SAR and SAR with Learning (SARL) on selected instances. The reinforcement learning used in SARL eliminates the manual setting of neighborhood structure composition as required in SAR. Furthermore, experimental results show that SARL is more effective compared to SAR. The way multiple operators are weighted (acceptance rate and CPU time) and selected (Roulette Wheel) improves the connectivity of search space and number of transitions per time unit.

We incorporated the average cost changes into the reheated temperature function on selected instances. In effect, exploration is guided by the gradient of the search landscape. In a comparison, SAIRL is better than SARL on selected instances in terms of average means of soft constraints violations. In fact, SAIRL is particularly effective for the instance ITC07-1.

We also compared the performance of SAIRL with SAR and other state of the art methods. SAIRL is comparable, or better, than SAR and other state of the art methods without the need for parameter tuning. In addition, we illustrated the movement of neighborhood structure composition. Finally, we have shown that SAIRL is extendable when the runtime is extended.

Chapter 8

Conclusion and Future Research

As a conclusion of the study, section 8.1 presents the summary of the research work carried out and highlights the contributions that can be drawn from this research. Section 8.2 outlines the possible future research work that is suggested to further develop ideas presented in this thesis.

8.1 Thesis Summary

In chapter 2, we reviewed the scientific literature on educational timetabling where key introduction and surveys are provided. There are three variants of educational timetabling identified, namely school, course and examination timetabling. We described how timetabling is closely associated with the graph coloring problem. Generally, the algorithms used to solve timetabling problems are either one-stage, multi-stage or multi-stage with relaxation. One of the most popular approaches utilized in educational timetabling problems is Meta-heuristics which can be categorized as local search based (SD, FD, TS, GD, SA, ILS, VNS) and evolutionary based (GA, ACO). Another popular approach is hyper-heuristics which aim to be generally applicable across optimization problems. In addition, we presented a review of MCTS for combinatorial optimization problems. MCTS has never been utilized for timetabling problems. Finally, we reviewed the implementation of hybrid algorithms.

In chapter 3, we described the requirements of the course timetabling problem. The datasets (Socha, ITC02, ITC07, Hard) considered in this thesis are described, including their origin, time limit, hard and soft constraints. In addition, the instance specific statistics such as the # of events, # of rooms, # of features and # of students are provided. The formal presentation of the problem is also presented. The related work for each dataset are reviewed in detail and presented in a chronological order. State of the art methods are identified and studied. Generally, the best performing methods are based on Simulated Annealing. However, as evident in the literature, the methodologies either require extensive parameter tuning to obtain good quality results or are only utilized on a limited number of datasets.

In chapter 4, we compare the effectiveness of several constructive heuristic approaches in finding feasible solutions for the course timetabling problem. We focus on MCTS as well as other state of the art methods. We compared several graph coloring heuristics (LD, SD, DSR) using two types of assignments, namely assignment by place and assignment by slot. We found that the variant based on slot assignment and DSR heuristic is the most effective in terms of average means of unassigned events for all the datasets considered. We compared random and heuristic based simulation (DSR) for the simulation part of MCTS. It seems heuristic based simulation is better than random simulation. Heuristics make simulation more realistic compared to random simulation where events and slots are selected randomly. We tested several tree pruning heuristics (DSR, LD-All, MV-All and SD-All). Tree pruning improves the effectiveness of MTCS in finding feasible solutions in terms of average number of unassigned events. Empirical results also show that MV-All worked best among the heuristics tested. We compared the best variant of GCH, MCTS and TS in finding feasible solutions. MCTS worked well for Socha and ITC02 instances but was lacking in terms of performance for ITC07 instances. MCTS could not find a feasible solution for instances 10 and 22 of ITC07 even with extended runtime. Computational experience shows that the value of B (selection part of MCTS) can affect the results. For MCTS to work effectively, this value needs to be adjusted for specific instances. In addition, the suitable value also depends on the *runtime*. Overall, TS shows great potential and therefore we decided to focus on improving the algorithm, which is our focus in the next chapter.

In a comparison with GCH and MCTS, TS is apparently the best method

in finding feasible solutions. Therefore, we focus on improving TS. In chapter 5, we propose several enhancements to the algorithm. We have presented the effect of sampling on TS. TS with event sampling is more effective than TS without event sampling. Event sampling size S=[0.25% $\times$ # of events], is more effective than event sampling size $S = 1$, particularly when the # of events is high. Event sampling size S=[0.25% $\times$ # of events] not only allows more moves per time unit but also naturally varies the diversification and intensification capability of the algorithm (depending on the the number of unassigned events) despite the fixed S throughout the search. We compared the effect of using different cost functions with or without perturbation on TS with sampling. The novel cost function proposed in Eq. 5.1 performed better than other cost functions used in the scientific literature regardless of whether perturbation is used. The results are further improved when the proposed perturbation is paired with the proposed cost function on TS with event sampling. The proposed cost function is exploitative, and the proposed perturbation is exploratory. A good balance between exploration (proposed perturbation) and exploitation (proposed cost function) for the search is achieved when the perturbation is initiated at the right time. When perturbation is called too early (more frequent), the search lacks exploitation. Meanwhile if perturbation is initiated too late (less frequent), the search lacks exploration. Overall, TSSP is shown to be more effective in finding feasible solutions for the benchmark timetabling problem compared to TS. The number of unassigned events and average time to feasibility are presented for all the datasets. In addition, t-tests are conducted to compare the means for these values between TS and TSSP. TSSP managed to find 100% feasibility for all Socha, ITC02 and ITC07 instances in relatively short time compared to existing methods in the scientific literature. TSSP is further enhanced by hybridization with ILS. We compared the effectiveness of TSSP and TSSP-ILS in finding feasible solutions for considerably hard benchmarked timetabling problems. The method is not only superior to the performance of either TSSP or ILS alone but also other state of the art methods. In addition, significant improvement of result is observed when the runtime is extended suggesting that TSSP-ILS is extendable. As we have developed an efficient algorithm in finding feasible solutions, we shift our focus on improving the soft constraint violations of the feasible solutions in the next chapter.

In chapter 6, we improve the feasible solutions in terms of soft constraint violations by using a method based on SA. We focus on SA as it has been

very effective in addressing combinatorial optimization problems, particularly timetabling problems. In fact, all state of the art methods for the instances considered in this work are based on SA. We have compared the effectiveness of both individual and combination of neighborhood structures. The right combination of neighborhood structures is important in getting high quality solutions. It is more effective than any individual operator used alone. However, finding the right ratio for the operators is difficult as its effectiveness is dependent on the instances. We analyzed various neighborhood examination schemes or the method of selecting events and time slots for move evaluation. We found that DE-DS (deterministic event and deterministic slot) is more effective than both RE-RS (random event and random slot) and DE-RS (deterministic event and random slot). DE-DS allows the search space to be thoroughly explored. We presented the effect of basic reheating. The temperature is reheated to a value as a function of the current cost when the search is assumed stuck. In effect, exploration is guided by the current cost. The setup with reheating generated better results than its counterpart without reheating. We tested three types of local optima detection. Type 2 (relative changes of current cost) seems to be more effective than Type 1 (absolute changes of current cost) and Type 3 (relative changes of current cost). A precise local optima estimator is vital to prevent unnecessary reheating which may affect the exploitation capability of the search. We also compared the effect of incremental reheating where temperature is reheated to a higher value if the search is still stuck since the previous reheating. Slightly better results are observed when incremental reheating is used on the selected instances. Experimental results show that incremental reheating thrives while basic reheating suffers in terms of performance when the current cost is low. Incremental reheating helps the search to escape from local optima while ensuring that the current solution does not stray too much (preserving previous search effort). Finally, we compared the performance of the SAR algorithm with state of the art methods. Competitive results in terms of soft constraint violations are reported in all datasets tested. The behaviour of SAR and the effect of reheating are also displayed. Moreover, SAR is also shown to be extendable when the runtime is extended.

In chapter 7, we further enhance the SAR algorithm in terms of ease of use and performance (soft constraint violations). We have compared the results of SAR and SAR with Learning (SARL) on selected instances. The reinforcement learning used in SARL eliminates the manual setting of neighborhood struc-

ture composition as required in SAR. Furthermore, experimental results show that SARL is more effective compared to SAR. The way multiple operators are weighted (acceptance rate and CPU time) and selected (Roulette Wheel) improves the connectivity of search space and number of transitions per time unit. We incorporated the average cost changes into the reheated temperature function on selected instances. In effect, exploration is guided by the gradient of the search landscape. In a comparison, SAIRL is better than SARL on selected instances in terms of average means of soft constraints violations. In fact, SAIRL is particularly effective for the instance ITC07-1. We also compared the performance of SAIRL with SAR and other state of the art methods. SAIRL is comparable, or better, than SAR and other state of the art methods without the need for parameter tuning. In addition, we illustrated the movement of neighborhood structure composition. Finally, we have shown that SAIRL is extendable when the runtime is extended.

8.2 Future Research

It would be interesting to see how TSSP-ILS performs on graph coloring problem as the algorithm is based on PARTIALCOL which was initially used to tackle graph coloring. We expect to see improved results similar to the improvements made by TSSP-ILS on course timetabling problems.

We plan to utilize SAIRL on other educational timetabling problems such as school timetabling (ITC11) and examination timetabling (examination track of ITC07 and Toronto dataset). Adaptations should be minimal considering the common features and structures shared by the educational timetabling problems. The algorithm could also be applied to other scheduling problems (transport scheduling, sports scheduling and nurse rostering) and possibly other combinatorial optimization problems (bin packing, vehicle routing). Working on different problems not only provides a platform to test the robustness and general applicability of SAIRL but also provides further evidence which might lead to further algorithm enhancements.

It is also possible to add more complex operators into the neighborhood structure compostion such as Hungarian method [111], double Kempe [124] etc. The complex operators may be computationally expensive but are worthwhile

provided the connectivity of the search space is improved. The reinforcement learning used in SAIRL will adjust to the neighborhood structure composition accordingly based on the acceptance ratio and computational cost (CPU time) of the operators.

SAIRL can be hybridized with Tabu Search. In SAIRL, every time slot is attempted for each event unless the event is successfully moved to a time slot. An event can be prohibited from moving to certain slots after moving out of the time slots recently. Instead of attempting to move an event to every time slot, only certain non-tabu time slots would be attempted. It is hoped that the exploration of the search will improve. Tabu tenure determines the prohibition duration (number of iterations) of a particular time slot for a particular event. We could set the tabu tenure as a function proportional to the current cost. This dynamic tabu tenure is expected to allow the search to explore and exploit the search space accordingly during the search process. This idea is inspired by the principle that the search should explore more when the current cost is high and exploit more when the current cost is low.

SAIRL can also be hybridized with any population based algorithms (GA) which are well known for their exploration capability. For instance, when SAIRL is stuck during the search, GA can be initiated for the search to escape from being stuck in a local optima. We would suggest that the number iterations for the genetic algorithm to be set proportional to the current cost. After a certain number of iterations, the mode of execution is returned back to SAIRL. Then the temperature is set proportional to the current cost and cooled until it is stuck again. The execution of SAIRL and GA are alternated until the time limit is reached.

Currently, SAIRL is utilizing a static cooling schedule. We are looking at testing SAIRL with various adaptive cooling schedules as proposed in [178], [149], [137] and [172]. The adaptive cooling schedules worked well for the respective domains. However, a parameter value has to be set for them to work effectively.

Bibliography

[1] Emile Aarts, Jan Korst, and Wil Michiels. Simulated annealing. In Edmund K. Burke and Graham Kendall, editors, *Search Methodologies*, pages 265–285. Springer US, 2014.

[2] Syariza Abdul Rahman, Andrzej Bargiela, Edmund K. Burke, Ender Ã–zcan, Barry McCollum, and Paul McMullan. Adaptive linear combination of heuristic orderings in constructing examination timetables. *European Journal of Op*, 232(2):287–297, 2014.

[3] Salwani Abdullah, Edmund K Burke, and Barry McCollum. An investigation of variable neighbourhood search for university course timetabling. In *The 2nd Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA)*, pages 413–427, 2005.

[4] Salwani Abdullah, Edmund K Burke, and Barry McCollum. A hybrid evolutionary approach to the university course timetabling problem. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 1764–1768. IEEE, 2007.

[5] Salwani Abdullah, Edmund K Burke, and Barry McCollum. Using a randomised iterative improvement algorithm with composite neighbourhood structures for the university course timetabling problem. In *Metaheuristics*, pages 153–169. Springer, 2007.

[6] Salwani Abdullah, Laleh Golafshan, and Mohd Zakree Ahmad Nazri. Reheat simulated annealing algorithm for rough set attribute reduction. *International Journal of Physical Sciences*, 6(8):2083–2089, 2011.

[7] Salwani Abdullah, Khalid Shaker, Barry McCollum, and Paul McMullan. Construction of course timetables based on great deluge and tabu search. In *Metaheuristics Int. Conf., VIII Meteheuristic*, pages 13–16, 2009.

[8] Salwani Abdullah and Hamza Turabieh. Generating university course timetable using genetic algorithms and local search. In *Convergence and Hybrid Information Technology, 2008. ICCIT'08. Third International Conference on*, volume 1, pages 254–260. IEEE, 2008.

[9] David Abramson. Constructing school timetables using simulated annealing: sequential and parallel algorithms. *Management science*, 37(1):98–113, 1991.

[10] David Abramson, Mohan Krishna Amoorthy, and Henry Dang. Simulated annealing cooling schedules for the school timetabling problem. *Asia-Pacific Journal of Operational Research*, 16(1):1, 1999.

[11] Anmar Abuhamdah, Masri Ayob, Graham Kendall, and Nasser R Sabar. Population based local search for university course timetabling problems. *Applied Intelligence*, 40(1):44–53, 2014.

[12] Mohammed Azmi Al-Betar and Ahamad Tajudin Khader. A harmony search algorithm for university course timetabling. *Annals of Operations Research*, 194(1):3–31, 2012.

[13] Fraser Alex. Simulation of genetic systems by automatic digital computers. i. introduction. *Aust. J. Biol. Sci*, 10:484–491, 1957.

[14] B. Arneson, R. B. Hayward, and P. Henderson. Monte carlo tree search in hex. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(4):251–258, 2010.

[15] Halvard Arntzen and Arne Lokketangen. A local search heuristic for a university timetabling problem. *nine*, 1(T2):T45, 2003.

[16] Hishammuddin Asmuni, Edmund K Burke, and Jonathan M Garibaldi. Fuzzy multiple heuristic ordering for course timetabling. In *Proceeding of the 5th United Kingdom Workshop on Computational Intelligence, London*, pages 302–309. Citeseer, 2005.

[17] Nader Azizi and Saeed Zolfaghari. Adaptive temperature control for simulated annealing: a comparative study. *Computers and Operations Research*, 31(14):2439–2451, 2004.

[18] R. Bai, E. K. Burke, M. Gendreau, G. Kendall, and B. McCollum. Memory length in hyper-heuristics: An empirical study. In *2007 IEEE Symposium on Computational Intelligence in Scheduling*, pages 173–178, 2007.

[19] Ruibin Bai, Edmund K. Burke, Graham Kendall, Jingpeng Li, and Barry McCollum. A Hybrid Evolutionary Approach to the Nurse Rostering Problem. *IEEE Transactions on Evolutionary Computation*, 14(4):580–590, AUG 2010.

[20] Victor A. Bardadym. Computer-aided school and university timetabling: The new wave. In Edmund Burke and Peter Ross, editors, *Practice and Theory of Automated Timetabling*, Lecture Notes in Computer Science, pages 22–45. Springer Berlin Heidelberg, 1995.

[21] Michele Battistutta, Andrea Schaerf, and Tommaso Urli. Feature-based tuning of single-stage simulated annealing for examination timetabling. *Annals of Operations Research*, pages 1–16, 2015.

[22] Ivo Blochliger and Nicolas Zufferey. A graph coloring heuristic using partial solutions and a reactive tabu scheme. *Computers & Operations Research*, 35(3):960–975, 2008.

[23] Daniel Brelaz. New methods to color the vertices of a graph. *Commun. (ACM)*, 22(4):251–256, 1979.

[24] Hans J Bremermann. *The evolution of intelligence: The nervous system as a model of its environment.* University of Washington, Department of Mathematics, 1958.

[25] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(1):1–43, 2012.

[26] E. K. Burke, D. G. Elliman, and R. Weare. A university timetabling system based on graph colouring and constraint manipulation. *Journal of Research on Computing in Education*, 27(1):1–18, 1994.

[27] Edmund Burke, Yuri Bykov, James Newall, and Sanja Petrovic. A time-predefined approach to course timetabling. *The Yugoslav Journal of Operations Research ISSN: 0354-0243 EISSN: 2334-6043*, 13(2), 2003.

[28] Edmund Burke, Dave Elliman, Peter Ford, and Rupert Weare. Examination timetabling in british universities: A survey. In Edmund Burke and Peter Ross, editors, *Practice and Theory of Automated Timetabling*, Lecture Notes in Computer Science, pages 76–90. Springer Berlin Heidelberg, 1995.

[29] Edmund Burke, Kirk Jackson, Jeffrey H. Kingston, and Rupert Weare. Automated university timetabling: The state of the art. *The computer journal*, 40(9):565–571, 1997.

[30] Edmund Burke, Graham Kendall, Jim Newall, Emma Hart, Peter Ross, and Sonia Schulenburg. Hyper-heuristics: An emerging direction in modern search technology. In Fred Glover and Gary A. Kochenberger, editors, *Handbook of Metaheuristics*, number 57 in International Series in Operations Research & Management Science, pages 457–474. Springer US, 2003.

[31] Edmund K Burke, Patrick De Causmaecker, Greet Vanden Berghe, and Hendrik Van Landeghem. The state of the art of nurse rostering. *Journal of scheduling*, 7(6):441–499, 2004.

[32] Edmund K. Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Ã–zcan, and Rong Qu. Hyper-heuristics: a survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013.

[33] Edmund K Burke and Graham Kendall. *Search methodologies: introductory tutorials in optimization and decision support techniques*. Springer, 2005.

[34] Edmund K. Burke, Graham Kendall, Mustafa Misir, and Ender Ozcan. Monte carlo hyper-heuristics for examination timetabling. *Annals of Operations Research*, 196(1):73–90, 2012.

[35] Edmund K Burke, Graham Kendall, and Eric Soubeiga. A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics*, 9(6):451–470, 2003.

[36] Edmund K Burke, Barry McCollum, Amnon Meisels, Sanja Petrovic, and Rong Qu. A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research*, 176(1):177–192, 2007.

[37] Edmund K Burke and James P Newall. Enhancing timetable solutions with local search methods. In *Practice and Theory of Automated Timetabling IV*, pages 195–206. Springer, 2003.

[38] Edmund Kieran Burke and Sanja Petrovic. Recent research directions in automated timetabling. *European Journal of Operational Research*, 140(2):266–280, 2002.

[39] EK Burke and JD Landa Silva. The design of memetic algorithms for scheduling and timetabling problems. In *Recent Advances in Memetic Algorithms*, pages 289–311. Springer, 2005.

[40] Yuri Bykov and Sanja Petrovic. An initial study of a novel step counting hill climbing heuristic applied to timetabling problems. In *Mista2013*, 2013.

[41] Hadrien Cambazard, Emmanuel Hebrard, Barry O'Sullivan, and Alexandre Papadopoulos. Local search and constraint programming for the post enrolment-based course timetabling problem. *Annals of Operations Research*, 194(1):111–135, 2012.

[42] A. Caprara, M. Fischetti, P. L. Guida, M. Monaci, G. Sacco, and P. Toth. Solution of real-world train timetabling problems. In *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*, pages 10 pp.–, 2001.

[43] Alberto Caprara, Matteo Fischetti, and Paolo Toth. Modeling and solving the train timetabling problem. *Operations*, 50(5):851–861, 2002.

[44] Michael W. Carter. A survey of practical applications of examination timetabling algorithms. *Operations Research*, 34(2):193–202, 1986.

[45] Michael W Carter and DG Johnson. Extended clique initialisation in examination timetabling. *Journal of the operational research society*, pages 538–544, 2001.

[46] Michael W. Carter and Gilbert Laporte. Recent developments in practical examination timetabling. In Edmund Burke and Peter Ross, editors, *Practice and Theory of Automated Timetabling*, Lecture Notes in Computer Science, pages 1–21. Springer Berlin Heidelberg, 1995.

[47] Michael W. Carter and Gilbert Laporte. Recent developments in practical course timetabling. In *International Conference on the Practice and Theory of Automated Timetabling*, pages 3–19. Springer, 1997.

[48] Sara Ceschia, Luca Di Gaspero, and Andrea Schaerf. Design, engineering, and experimental analysis of a simulated annealing approach to the post-enrolment course timetabling problem. *Computers & Operations Research*, 39(7):1615–1624, 2012.

[49] GMJB Chaslot, Steven De Jong, Jahn-Takeshi Saito, and JWHM Uiterwijk. Monte-carlo tree search in production management problems. In *Proceedings of the 18th BeNeLux Conference on Artificial Intelligence*, pages 91–98. Citeseer, 2006.

[50] Guillaume Chaslot. *Monte-carlo tree search*. PhD thesis, PhD thesis, Maastricht University, 2010.

[51] Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte-carlo tree search: A new framework for game AI. In *AIIDE*, 2008.

[52] Marco Chiarandini, Mauro Birattari, Krzysztof Socha, and Olivia Rossi-Doria. An effective hybrid algorithm for university course timetabling. *Journal of Scheduling*, 9(5):403–432, 2006.

[53] Marco Chiarandini, Chris Fawcett, and Holger H Hoos. A modular multiphase heuristic solver for post enrollment course timetabling. In *Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2008)*, 2008.

[54] Tim B Cooper and Jeffrey H Kingston. *The complexity of timetable construction problems*. Springer, 1996.

[55] Jean Franois Cordeau, Brigitte Jaumard, and Rodrigo Morales. Efficient timetabling solution with tabu search. *International Timetabling Competition*, 2003.

[56] Daniele Costa and Alain Hertz. Ants can colour graphs. *Journal of the operational research society*, 48(3):295–305, 1997.

[57] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *Computers and games*, pages 72–83. Springer, 2007.

[58] RÃ©mi Coulom. Monte-carlo tree search in crazy stone. In Takeshi Ito and Akihiro Kishimoto, editors, *12th Game Programming Workshop*, 2007.

[59] S Daskalaki, T Birbas, and E Housos. An integer programming formulation for a case study in university timetabling. *European Journal of Ope*, 153(1):117–135, 2004.

[60] D. de Werra, A. S. Asratian, and S. Durand. Complexity of some special types of timetabling problems. *Journal of Scheduling*, 5(2):171–183, 2002.

[61] Dominique de Werra. An introduction to timetabling. *European Journal of Operational Research*, 19(2):151–162, 1985.

[62] Luca Di Gaspero and Andrea Schaerf. Multi-neighbourhood local search with application to course timetabling. In *Practice and Theory of Automated Timetabling IV*, pages 262–275. Springer, 2003.

[63] Luca Di Gaspero and Andrea Schaerf. Timetabling competition ttcomp 2002: solver description. *International Timetabling Competition*, 2003.

[64] Marco Dorigo, Vittorio Maniezzo, Alberto Colorni, and Vittorio Maniezzo. Positive feedback as a search strategy. *Technical report 91-016, Politecnico di Milano, Italy*, 1991.

[65] Raphaël Dorne and Jin-Kao Hao. A new genetic local search algorithm for graph coloring. In *Parallel Problem Solving from Nature—PPSN V*, pages 745–754. Springer, 1998.

[66] K Dowsland, IH Osman, and JP Kelly. Metaheuristics: theory & applications. *Metaheuristics: Theory and Applications*, 1996.

[67] KA Dowsland and JM Thompson. Ant colony optimization for the examination scheduling problem. *Journal of the Operational Research Society*, 56(4):426–438, 2005.

[68] Kathryn A Dowsland and Jonathan M Thompson. An improved ant colony optimisation heuristic for graph colouring. *Discrete Applied Mathematics*, 156(3):313–324, 2008.

[69] Peter Drake and Steve Uurtamo. Move ordering vs heavy playouts: Where should heuristics be applied in monte carlo go? In *Proceedings of the 3rd North American Game-On Conference*, 2007.

[70] Gunter Dueck. New optimization heuristics: the great deluge algorithm and the record-to-record travel. *Journal of Computational physics*, 104(1):86–92, 1993.

[71] Naveed Ejaz and Muhammad Younus Javed. A hybrid approach for course scheduling inspired by die-hard co-operative ant behavior. In *Automation and Logistics, 2007 IEEE International Conference on*, pages 3095–3100. IEEE, 2007.

[72] M. A. Saleh Elmohamed, Paul Coddington, and Geoffrey Fox. A comparison of annealing techniques for academic course scheduling. In Edmund Burke and Michael Carter, editors, *Practice and Theory of Automated Timetabling II*, Lecture Notes in Computer Science, pages 92–112. Springer Berlin Heidelberg, 1997.

[73] M. Enzenberger, M. Muller, B. Arneson, and R. Segal. Fuego-an open-source framework for board games and go engine based on monte carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(4):259–270, 2010.

[74] Maximilian M. Etschmaier and Dennis F. X. Mathaisel. Airline scheduling: An overview. *Transportation Science*, 19(2):127–138, 1985.

[75] S. Even, A. Itai, and A. Shamir. On the complexity of time table and multi-commodity flow problems. In *16th Annual Symposium on Foundations of Computer Science (sfcs 1975)*, pages 184–193, 1975.

[76] E Falkenauer. A new representation and operators for gas applied to grouping problems (1994). *Evolutionary Computation*, 2(2), 1994.

[77] Emanuel Falkenauer. *Genetic algorithms and grouping problems*. John Wiley & Sons, Inc., 1998.

[78] Charles Fleurent and Jacques A Ferland. Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research*, 63(3):437–461, 1996.

[79] G. H. G. Fonseca, H. G. Santos, T. A. M. Toffolo, S. S. Brito, and M. J. F. Souza. A sa-ils approach for the high school timetabling problem. In *Proceedings of the ninth international conference on the practice and theory of automated timetabling (PATAT 2012)*, pages 493–496. Citeseer, 2012.

[80] Philippe Galinier and Jin-Kao Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of combinatorial optimization*, 3(4):379–397, 1999.

[81] Luca Di Gaspero and Andrea Schaerf. Neighborhood portfolio approach for local search applied to timetabling problems. *Journal of Mathematical Modelling and Algorithms*, 5(1):65–89, 2006.

[82] Sylvain Gelly and David Silver. Monte-carlo tree search and rapid action value estimation in computer go. *Artificial Intelligence*, 175(11):1856–1875, 2011.

[83] Sylvain Gelly, Yizao Wang, Rémi Munos, Olivier Teytaud, et al. Modification of uct with patterns in monte-carlo go. *Diss. INRIA*, 2006.

[84] Michel Gendreau and Jean-Yves Potvin. Tabu search. In Edmund K. Burke and Graham Kendall, editors, *Search Methodologies*, pages 243–263. Springer US, 2014.

[85] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5):533–549, 1986.

[86] Fred Glover and Manuel Laguna. Tabu search. In Panos M. Pardalos, Ding-Zhu Du, and Ronald L. Graham, editors, *Handbook of Combinatorial Optimization*, pages 3261–3362. Springer New York, 2013.

[87] Christos Gogos, Panayiotis Alefragis, and Efthymios Housos. An improved multi-staged algorithmic process for the solution of the examination timetabling problem. *Annals of Operations Research*, 194(1):203–221, 2012.

[88] Say Leng Goh, Graham Kendall, and Nasser R. Sabar. Improved local search approaches to solve post enrolment course timetabling problem. *European Journal of Operational Research*, 2017.

[89] David E Golberg. Genetic algorithms in search, optimization, and machine learning. *Addion wesley*, 1989, 1989.

[90] Bruce L. Golden, S. Raghavan, and Edward A. Wasil. *The Vehicle Routing Problem: Latest Advances and New Challenges*. Springer Science & Business Media, 2008.

[91] R. L. Graham and P. Hell. On the history of the minimum spanning tree problem. *Annals of the History of Computing*, 7(1):43–57, 1985.

[92] S.R.A. Haddadene, N. Labadie, and C. Prodhon. A GRASP x ILS for the vehicle routing problem with time windows, synchronization and precedence constraints. *Expert Systems with Applications*, 66:274–294, 2016.

[93] Bruce Hajek. Cooling schedules for optimal annealing. *Mathematics of Operations Research*, 13(2):311–329, 1988.

[94] Pierre Hansen and Nenad Mladenovic. Variable neighborhood search for the p-median. *Location Science*, 5(4):207–226, 1997.

[95] Pierre Hansen and Nenad Mladenovic. Variable neighborhood search. In Edmund K. Burke and Graham Kendall, editors, *Search Methodologies*, pages 313–337. Springer US, 2014.

[96] S. He, Y. Wang, F. Xie, J. Meng, H. Chen, S. Luo, Z. Liu, and Q. Zhu. Game player strategy pattern recognition and how UCT algorithms apply pre-knowledge of player's strategy to improve opponent AI. In *2008 International Conference on Computational Intelligence for Modelling Control Automation*, pages 1177–1181, 2008.

[97] Alain Hertz and Dominique de Werra. Using tabu search techniques for graph coloring. *Computing*, 39(4):345–351, 1987.

[98] Karla L. Hoffman, Manfred Padberg, and Giovanni Rinaldi. Traveling salesman problem. In Saul I. Gass and Michael C. Fu, editors, *Encyclopedia of Operations Research and Management Science*, pages 1573–1578. Springer US, 2013.

[99] JH Holland. Adaption in natural and artificial systems. *Ann Arbor MI: The University of Michigan Press*, 1975.

[100] J. Huang, Z. Liu, B. Lu, and F. Xiao. Pruning in UCT algorithm. In *2010 International Conference on Technologies and Applications of Artificial Intelligence*, pages 177–181, 2010.

[101] Omar J. Ibarra-Rojas and Yasmin A. Rios-Solis. Synchronization of bus timetabling. *Transportation Research Part B: Methodological*, 46(5):599–614, 2012.

[102] Ghaith M Jaradat and Masri Ayob. An elitist-ant system for solving the post-enrolment course timetabling problem. In *Database Theory and*

Application, Bio-Science and Bio-Technology, pages 167–176. Springer, 2010.

[103] David S Johnson. The NP-completeness column: an ongoing guide. *Journal of Algorithms*, 6(3):434–451, 1985.

[104] MN Mohmad Kahar and Graham Kendall. A great deluge algorithm for a real-world examination timetabling problem. *Journal of the Operational Research Society*, 66(1):116–133, 2013.

[105] Graham Kendall, Sigrid Knust, Celso C Ribeiro, and Sebastián Urrutia. Scheduling in sports: An annotated bibliography. *Computers & Operations Research*, 37(1):1–19, 2010.

[106] Jeffrey H. Kingston. Educational timetabling. In A. Sima Uyar, Ender Ozcan, and Neil Urquhart, editors, *Automated Scheduling and Planning*, number 505 in Studies in Computational Intelligence, pages 91–108. Springer Berlin Heidelberg, 2013.

[107] Scott Kirkpatrick, C Daniel Gelatt, Mario P Vecchi, et al. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[108] Levente Kocsis and Csaba Szepesvari. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006*, pages 282–293. Springer, 2006.

[109] Philipp Kostuch. Timetabling competition-sa-based heuristic. *International Timetabling Competition*, 2003.

[110] Philipp Kostuch. The university course timetabling problem with a three-phase approach. In *Practice and Theory of Automated Timetabling V*, pages 109–125. Springer, 2005.

[111] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1):83–97, 1955.

[112] Vipin Kumar. Algorithms for constraint-satisfaction problems: A survey. *AI Magazine*, 13(1):32, 1992.

[113] Dario Landa-Silva and Joe Henry Obit. Great deluge with non-linear decay rate for solving course timetabling problems. In *Intelligent Systems, 2008. IS'08. 4th International IEEE Conference*, volume 1, pages 8–11. IEEE, 2008.

[114] Dario Landa-Silva and Joe Henry Obit. Evolutionary non-linear great deluge for university course timetabling. In *Hybrid Artificial Intelligence Systems*, pages 269–276. Springer, 2009.

[115] C. S. Lee, M. H. Wang, G. Chaslot, J. B. Hoock, A. Rimmel, O. Teytaud, S. R. Tsai, S. C. Hsu, and T. P. Hong. The computational intelligence of MoGo revealed in taiwan's computer go tournaments. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(1):73–89, 2009.

[116] C. S. Lee, M. H. Wang, S. J. Yen, T. H. Wei, I. C. Wu, P. C. Chou, C. H. Chou, M. W. Wang, and T. H. Yan. Human vs. computer go: Review and prospect [discussion forum]. *IEEE Computational Intelligence Magazine*, 11(3):67–72, 2016.

[117] Rhyd Lewis. A time-dependent metaheuristic algorithm for post enrolment-based course timetabling. *Annals of Operations Research*, 194(1):273–289, 2012.

[118] Rhyd Lewis and Jonathan Thompson. Analysing the effects of solution space connectivity with an effective metaheuristic for the course timetabling problem. *European Journal of Operational Research*, 240(3):637–648, 2015.

[119] Rhydian Lewis. A survey of metaheuristic-based techniques for university timetabling problems. *OR spectrum*, 30(1):167–190, 2008.

[120] Rhydian Lewis and Ben Paechter. Finding feasible timetables using group-based operators. *Evolutionary Computation, IEEE Transactions on*, 11(3):397–413, 2007.

[121] Yongkai Liu, Defu Zhang, and Francis YL Chin. A clique-based algorithm for constructing feasible timetables. *Optimization Methods & Software*, 26(2):281–294, 2011.

[122] P. Lopez-Garcia, E. Onieva, E. Osaba, A. D. Masegosa, and A. Perallos. GACE: A meta-heuristic based in the hybridization of Genetic Algorithms and Cross Entropy methods for continuous optimization. *Expert Systems with Applications*, 55:508–519, AUG 15 2016.

[123] Helena R. Lourenco, Olivier C. Martin, and Thomas Stutzle. Iterated local search. In Fred Glover and Gary A. Kochenberger, editors, *Handbook of*

Metaheuristics, number 57 in International Series in Operations Research & Management Science, pages 320–353. Springer US, 2003.

[124] Zhipeng Lu and Jin-Kao Hao. Adaptive tabu search for course timetabling. *European Journal of Operational Research*, 200(1):235–244, 2010.

[125] Silvano Martello, David Pisinger, and Daniele Vigo. The three-dimensional bin packing problem. *Operations Research*, 48(2):256–267, 2000.

[126] S Matsumoto, N Hirosue, K Itonaga, N Ueno, and H Ishii. Monte-carlo tree search for a reentrant scheduling problem. In *Computers and Industrial Engineering (CIE), 2010 40th International Conference on*, pages 1–6. IEEE, 2010.

[127] Barry McCollum. A perspective on bridging the gap between theory and practice in university timetabling. In Edmund K. Burke and Hana RudovÃ¡, editors, *Practice and Theory of Automated Timetabling VI*, Lecture Notes in Computer Science, pages 3–23. Springer Berlin Heidelberg, 2006.

[128] Barry McCollum, Andrea Schaerf, Ben Paechter, Paul McMullan, Rhyd Lewis, Andrew J. Parkes, Luca Di Gaspero, Rong Qu, and Edmund K. Burke. Setting the research agenda in automated timetabling: The second international timetabling competition. *INFORMS Journal on Computing*, 22(1):120–130, 2009.

[129] Paul McMullan. An extended implementation of the great deluge algorithm for course timetabling. In *Computational Science–ICCS 2007*, pages 538–545. Springer, 2007.

[130] Daniel Merkle and Martin Middendorf. Swarm intelligence. In Edmund K. Burke and Graham Kendall, editors, *Search Methodologies*, pages 213–242. Springer US, 2014.

[131] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

[132] Pablo Moscato. Memetic algorithms: A short introduction. In *New ideas in optimization*, pages 219–234. McGraw-Hill Ltd., UK, 1999.

[133] Pablo Moscato et al. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech concurrent computation program, C3P Report*, 826:1989, 1989.

[134] Tomas Muller. Itc2007 solver description: A hybrid approach. *Annals of Operations Research*, 172(1):429–446, 2009.

[135] Clemens Nothegger, Alfred Mayer, Andreas Chwatal, and Günther R Raidl. Solving the post enrolment course timetabling problem by ant colony optimization. *Annals of Operations Research*, 194(1):325–339, 2012.

[136] J Obit, Dario Landa-Silva, Djamila Ouelhadj, and Marc Sevaux. Nonlinear great deluge with learning mechanism for solving the course timetabling problem. In *8th Metaheuristics International Conference (MIC 2009)*, 2009.

[137] Ralph HJM Otten and Lukas PPP van Ginneken. Floorplan design using annealing. In *The Best of ICCAD*, pages 479–488. Springer, 2003.

[138] Ender Ozcan, Burak Bilgin, and Emin Erkan Korkmaz. A comprehensive analysis of hyper-heuristics. *Intelligent Data Analysis*, 12(1):3–23, 2008.

[139] D. Perez, E. J. Powley, D. Whitehouse, P. Rohlfshagen, S. Samothrakis, P. I. Cowling, and S. M. Lucas. Solving the physical traveling salesman problem: Tree search and macro actions. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(1):31–45, 2014.

[140] Diego Perez, Philipp Rohlfshagen, and Simon M. Lucas. Monte-carlo tree search for the physical travelling salesman problem. In *Applications of Evolutionary Computation*, Lecture Notes in Computer Science, pages 255–264. Springer Berlin Heidelberg, 2012.

[141] Sanja Petrovic and Edmund K Burke. University timetabling. *Handbook of scheduling: algorithms, models, and performance analysis*, 45:1–23, 2004.

[142] Nelishia Pillay. An overview of school timetabling research. In *Proceedings of the international conference on the theory and practice of automated timetabling*, page 321, 2010.

[143] Gerhard Post, Luca Di Gaspero, Jeffrey H. Kingston, Barry McCollum, and Andrea Schaerf. The third international timetabling competition. *Annals of Operations Research*, 239(1):69–75, 2016.

[144] E. J. Powley, D. Whitehouse, and P. I. Cowling. Monte carlo tree search with macro-actions and heuristic route planning for the physical travelling salesman problem. In *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 234–241, 2012.

[145] R. Qu, E. K. Burke, B. McCollum, L. T. G. Merlot, and S. Y. Lee. A survey of search methodologies and automated system development for examination timetabling. *Journal of S*, 12(1):55–89, 2009.

[146] Rong Qu, Nam Pham, Ruibin Bai, and Graham Kendall. Hybridising heuristics within an estimation distribution algorithm for examination timetabling. *Applied Intelligence*, 42(4):679–693, JUN 2015.

[147] Colin R Reeves et al. Modern heuristic techniques. *VJ Rayward-Smith, IH Osman, CR Reeves, GD Smith (eds.). Modern Heuristic Search Methods, Wiley, Chichester*, pages 1–25, 1996.

[148] Arpad Rimmel, Fabien Teytaud, and Tristan Cazenave. Optimization of the nested monte-carlo algorithm on the traveling salesman problem with time windows. In Cecilia Di Chio, Anthony Brabazon, Gianni A. Di Caro, Rolf Drechsler, Muddassar Farooq, JÃ¶rn Grahl, Gary Greenfield, Christian Prins, Juan Romero, Giovanni Squillero, Ernesto Tarantino, Andrea G. B. Tettamanzi, Neil Urquhart, and A. Åžima Uyar, editors, *Applications of Evolutionary Computation*, Lecture Notes in Computer Science, pages 501–510. Springer Berlin Heidelberg, 2011.

[149] F. Romeo, Vincentelli Ak Sangiovanni, and Md Huang. An efficient general cooling schedule for simulated annealing. In *Proceeding of IEEE International Conference on Computer Aided Design*, 1986.

[150] Peter Ross, Dave Corne, and Hsiao-Lan Fang. Improving evolutionary timetabling with delta evaluation and directed mutation. In Yuval Davidor, Hans-Paul Schwefel, and Reinhard Manner, editors, *Parallel Problem Solving from Nature PPSN III*, Lecture Notes in Computer Science, pages 556–565. Springer Berlin Heidelberg, 1994.

[151] Peter Ross, Emma Hart, and Dave Corne. Some observations about ga-based exam timetabling. In *Practice and Theory of Automated Timetabling II*, pages 115–129. Springer, 1998.

[152] Olivia Rossi-Doria and Ben Paechter. A memetic algorithm for university course timetabling. *Combinatorial Optimization*, page 56, 2004.

[153] Olivia Rossi-Doria, Michael Sampels, Mauro Birattari, Marco Chiarandini, Marco Dorigo, Luca M. Gambardella, Joshua Knowles, Max Manfrin, Monaldo Mastrolilli, Ben Paechter, Luis Paquete, and Thomas Stutzle. A comparison of the performance of different metaheuristics on the timetabling problem. In Edmund Burke and Patrick De Causmaecker, editors, *Practice and Theory of Automated Timetabling IV*, Lecture Notes in Computer Science, pages 329–351. Springer Berlin Heidelberg, 2002.

[154] Thomas Philip Runarsson, Marc Schoenauer, and Michèle Sebag. Pilot, rollout and monte carlo tree search methods for job shop scheduling. In *Learning and Intelligent Optimization*, pages 160–174. Springer, 2012.

[155] Nasser R Sabar, Masri Ayob, Graham Kendall, and Rong Qu. A honey-bee mating optimization algorithm for educational timetabling problems. *European Journal of Operational Research*, 216(3):533–543, 2012.

[156] Nasser R Sabar, Masri Ayob, Rong Qu, and Graham Kendall. A graph coloring constructive hyper-heuristic for examination timetabling problems. *Applied Intelligence*, 37(1):1–11, 2012.

[157] Kumara Sastry, David E. Goldberg, and Graham Kendall. Genetic algorithms. In Edmund K. Burke and Graham Kendall, editors, *Search Methodologies*, pages 93–117. Springer US, 2014.

[158] Maarten PD Schadd, Mark HM Winands, H Jaap Van Den Herik, Guillaume MJ-B Chaslot, and Jos WHM Uiterwijk. Single-player monte-carlo tree search. In *Computers and Games*, pages 1–12. Springer, 2008.

[159] Andrea Schaerf. Local search techniques for large high school timetabling problems. *systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 29(4):368–377, 1999.

[160] Andrea Schaerf. A survey of automated timetabling. *Artificial intelligence review*, 13(2):87–127, 1999.

[161] Khalid Shaker and Salwani Abdullah. Controlling multi algorithms using round robin for university course timetabling problem. In *Database Theory and Application, Bio-Science and Bio-Technology*, pages 47–55. Springer, 2010.

[162] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[163] David Silver and Gerald Tesauro. Monte-carlo simulation balancing. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 945–952. ACM, 2009.

[164] Krzysztof Socha, Joshua Knowles, and Michael Sampels. A max-min ant system for the university course timetabling problem. In *Ant algorithms*, pages 1–13. Springer, 2002.

[165] Krzysztof Socha, Michael Sampels, and Max Manfrin. Ant algorithms for the university course timetabling problem with regard to the state-of-the-art. In *Applications of evolutionary computing*, pages 334–345. Springer, 2003.

[166] Jorge A. Soria-Alcaraz, Gabriela Ochoa, Jerry Swan, Martin Carpio, Hector Puga, and Edmund K. Burke. Effective learning hyper-heuristics for the course timetabling problem. *European Journal of Operational Research*, 238(1):77–86, 2014.

[167] John Stuart and R. J. D. Rutherford. Medical student concentration during lectures. *The Lancet*, 312(8088):514–516, 1978.

[168] B. Suman and P. Kumar. A survey of simulated annealing as a tool for single and multiobjective optimization. *Journal of the Operational Research Society*, 57(10):1143–1160, 2006.

[169] Lisa Ann Taylor. *Local search methods for the post enrolment-based course timetabling problem.* PhD thesis, Cardiff University, 2013.

[170] Jonathan M Thompson and Kathryn A Dowsland. Variants of simulated annealing for the examination timetabling problem. *Annals of Operations research*, 63(1):105–128, 1996.

[171] Jonathan M Thompson and Kathryn A Dowsland. A robust simulated annealing based examination timetabling system. *Computers & Operations Research*, 25(7):637–648, 1998.

[172] E. Triki, Y. Collette, and P. Siarry. A theoretical study on the behavior of simulated annealing leading to a new cooling schedule. *European Journal of Operational Research*, 166(1):77–92, 1998.

[173] Mauritsius Tuga, Regina Berretta, and Alexandre Mendes. A hybrid simulated annealing with kempe chain neighborhood for the university timetabling problem. In *Computer and Information Science, 2007. ICIS 2007. 6th IEEE/ACIS International Conference on*, pages 400–405. IEEE, 2007.

[174] Hamza Turabieh and Salwani Abdullah. Incorporating tabu search into memetic approach for enrolment-based course timetabling problems. In *Data Mining and Optimization, 2009. DMO'09. 2nd Conference on*, pages 115–119. IEEE, 2009.

[175] Hamza Turabieh, Salwani Abdullah, and Barry Mccollum. Electromagnetism-like mechanism with force decay rate great deluge for the course timetabling problem. In *Rough Sets and Knowledge Technology*, pages 497–504. Springer, 2009.

[176] Hamza Turabieh, Salwani Abdullah, Barry McCollum, and Paul McMullan. Fish swarm intelligent algorithm for the course timetabling problem. In *Rough Set and Knowledge Technology*, pages 588–595. Springer, 2010.

[177] Christos Valouxis and Efthymios Housos. Constraint programming approach for school timetabling. *Computers and Operations Research*, 30(10):1555–1572, 2003.

[178] Peter J Van Laarhoven and Emile H Aarts. *Simulated annealing: theory and applications*, volume 37. Springer Science & Business Media, 1987.

[179] Alam Zeb, Mushtaq Khan, Nawar Khan, Adnan Tariq, Liaqat Ali, Farooque Azam, and Syed Husain Imran Jaffery. Hybridization of simulated

annealing with genetic algorithm for cell formation problem. *International Journal of Advanced Manufacturing Technology*, 86(5-8):2243–2254, SEP 2016.