A Parallel Hybrid Metaheuristic Approach for Timetabling

A Thesis

Submitted to the Faculty of Graduate Studies and Research

In Partial Fulfilment of the Requirements

For the Degree of

Doctor of Philosophy

in

Computer Science

University of Regina

By

Ali Ahmed Altohami Hmer

Regina, Saskatchewan

May, 2013

UNIVERSITY OF REGINA

FACULTY OF GRADUATE STUDIES AND RESEARCH

SUPERVISORY AND EXAMINING COMMITTEE

Ali Ahmed Altohami Hmer, candidate for the degree of Doctor of Philosophy in Computer Science**,** has presented a thesis titled, **A Parallel Hybrid Metaheuristic Approach for Timetabling,** in an oral examination held on April 18, 2013. The following committee members have found the thesis acceptable in form and content, and that the candidate demonstrated satisfactory knowledge of the subject material.

External Examiner:	*Dr. Behrouz Homayoun Far, University of Calgary
Supervisor:	Dr. Malek Mouhoub, Department of Computer Science
Committee Member:	Dr. Yiyu Yao, Department of Computer Science
Committee Member:	Dr. Doug Farenick, Department of Mathematics and Statistics
Committee Member:	Dr. Samira Sadaoui-Mouhoub, Department of Computer Science
Chair of Defense:	Dr. Dongyan Blachford, Faculty of Graduate Studies and Research

*Participated Via SKYPE

Abstract

University timetabling is a well-known problem and much research has been done about it. This is an interesting subject to study because neither modeling nor solving it is straightforward. It has different variants ranging from teaching assignment to course timetabling to examination timetabling. Constructing any of these problems as a conflict-free timetable is a challenging task encountered every year by many educational institutions and universities across the world.

Most of the current research in timetabling concentrates on exploring how the level of generalization of algorithms applied can be advanced, so that a broader range of problems can be addressed. In this thesis, we pay special attention to two of these timetabling problems, first, the teaching assignment and second, the examination timetabling in a university environment. Two different methodologies that make use of two generic solvers to solve the aforementioned problems are proposed.

We propose an extension approach to the backtracking with look-ahead technique to solve the teaching assignment problem that adopts weighted partial satisfaction of soft constraints. Determining the optimal solution for a teaching assignment problem is a challenging task. The optimized look-ahead backtracking method applied to the solution is proposed and discussed along with computational

results of evaluation tests conducted on datasets collected from the Computer Science department at the University of Regina.

A multi-phase parallel hybrid metaheuristics approach integrating Tabu Search, Hill Climbing, Simulated Annealing and a modified version of Extended Great Deluge algorithms using metaheuristics techniques is proposed in relation to solve Large Examination Timetabling Problem. We present a pre-processing phase that includes a number of pre-construction procedures that does collections ordering and detecting some of the hidden constraints for a fast easy construction. We also propose a range of experiments on benchmark problem instances conducted to evaluate the proposed approach. In this regard, our method was capable of producing reasonable results in comparison to the best results published so far on the well-known benchmark datasets.

Acknowledgments

It is a pleasure to thank everyone who played a role in making this thesis possible. I take much pleasure to express my profound gratitude to my supervisor Dr. Malek Mouhoub for his persistent and inspiring supervision who supervised my thesis work from beginning to end.

I also thank my Ph.D. supervisory committee members, Dr. S. Sadaoui, Dr. Y. Yao and Dr. Douglas Farenick for their valuable suggestions and guidance.

My cordial thanks go to Higher Education Ministry, Government of Libya, for their financial and travel support.

I am also thankful to all my fellow researchers and faculty members in the Department of Computer Science for their spontaneous cooperation and encouragement.

Finally, I would like to thank all my friends and family for all their support. In particular, I would like to thank my wife, Reem, for her invaluable support when I most needed it.

I dedicate this thesis to my late father, a gentle soul who always stood behind me and knew I would succeed. To my beloved mother whose selfless support and inspiration has always been with me at each and every step of my life.

Contents

List of Figures

List of Tables

Acronyms

AAAI	American Association for Artificial Intelligence
AC	Ant Colonies
ACM	Association for Computing Machinery
API	*Application Programming Interface*
ASP	Application Service Provider
AVG	Average
BSID	Best Solution Iteration Difference
CBR	Case-Based Reasoning
CD	Conflicts Dictionary
COP	Constraints Optimization Problem
CP	Constraint Programming
CPU	Central Processing Unit
CSP	Constraints Satisfaction Problem
CTT	Course Time Tabling
DCSP	Distributed Constraints Satisfaction problem
EGD	Extended Great Deluge
ES	Evolution Strategies
ETP	Exam Timetabling
FIFO	First In First Out
GA	Genetic Algorithm
GB	Giga Byta
GD	Great Deluge
GHH	graph-based hyper-heuristic
GRASP	greedy randomized adaptive search procedures
HC	Hill Climbing
HTML	Hyper Text Markup Language
IDS	Incremental Dynamic Search
IEEE	Institute of Electrical and Electronics Engineers
IIS	Internet Information Server
IP	Integer Programming
ITC	International Timetabling Competition
LIFO	Last In First Out
LINQ	Language Integrated Query
MA	Memetic Algorithm
MEGD	Modified Extended Great Deluge
MPI	Message Passing Interface
MVC	Model View Controller
NP	Non-Polynomial
RNA	Randomised Non-Ascendant
SA	Simulated Annealing
SAT	Satisfiability
SQL	Structured Query Language
SS	Scatter Search
TAP	Teaching Assignment Problem
TAPS	Teaching Assignment Problem Solver
TS	Tabu Search
TT	Time Tabling
UTP	University Timetabling Problem
VNS	Variable Neighbourhood Search
WCF	Windows Communication Foundation
WSDL	Web Services Description Language
XML	EXtensible Markup Language

This page was left blank intentionally

1. Introduction

1.1 Motivations

The field of constraint optimization has been developing rapidly during the last 50 years. It has become an important field in computer science. The constraint optimization framework is an advantageous and well-studied framework expressing many problems of interest in Artificial Intelligence and other areas of Computer Science. Many real-life problems can be expressed as a special case of the constraint satisfaction problem. Some examples are scheduling, configuration, hardware verification, graph problems, molecular biology, etc. The search space is often exponential because the problem is NP-complete. Therefore a number of different approaches to the problem have been proposed to reduce the search space and find a feasible solution in a reasonable time.

Every year most academic institutions face the problem of scheduling to design new timetables in three main areas; courses, teaching assignments and examinations. The complexity of the problem lies in two main factors: the scheduling size and structure of the variables involved in the problem (i.e. exams, instructors, rooms, timeslots, courses, and students), and the number of constraints taking into account the significance of each constraint. As a result, an approach to solve the problem for a particular institution may not be appropriate for another one. Solutions to

timetabling problems have been proposed for the last five decades. Different timetabling problems have different constraints.

Historically, in a university, courses and teaching assignments timetabling are usually prepared, constructed and produced by departments or faculties and then students may have to schedule accordingly so that they can adopt the proposed timetable while, in contrast, examinations are normally scheduled at the university level [38]. The main problem with scheduling examination at different levels is that this kind of scheduling can be perceived by some students as unsatisfactory due to some factors as examinations may be scheduled too close to each other which would be considered as unfair.

To avoid such issues, it is common, in many universities that a timetable that has been generated and used previously, and found to be satisfactory is recycled, refined and adopted again in the upcoming years or semesters and usually applies to the three types of timetabling. Insignificant adjustments may be needed to be made and this can be done manually or with the help of a decision based system. As the difficulty and complexity of the problem increases, due to a natural growth in the number of students, courses, examinations, rooms, resources and other constraints and factors, the need for an automated timetabling system that can produce feasible, high quality timetables in a timely manner is often considered.

Teaching assignment and examination timetabling are one of the typical examples of constraint programming application. It is a very resilient and desirable topic of research at the present time fascinating the awareness of academic researchers from all over the world. [51], [52], [36] and [165] have published surveys and summaries on numerous methods and approaches applied by researchers to solve timetabling problems since 1960s to the mid of 2000s. Surveys, also, have shown that various methods have effectively solved particular problems and some algorithms along with heuristics were stated to work well with certain datasets whereas others achieved better results when performed with different datasets.

These results are the source of initial ideas behind the main motivation of researching building generic solvers using known algorithms, hybrid metaheuristics to explore the boundaries and limits of the generalization in solving timetabling problems. Hybrid metaheuristics are generally used as extensions to algorithms in certain situations to intelligently choose the right heuristic to solve classes of problems rather than solving just one problem [32].

This thesis was motivated by the prospect to construct an optimized parallel algorithm along with hybrid metaheuristics which should be able to solve a particular realistic problem with acceptable results although it is difficult to make a precise distinction between acceptable and not-acceptable timetables. This is because there is a large diversity in acceptance criteria due to the fact that realistic timetable

construction problems are multi-aspects based. Each aspect may introduce its own characteristics that add to the complexity of the problem. Therefore, only heuristic oriented solutions approaches are essentially feasible.

There is another motivation that is related to another aspect of COPs. In most real-life situations, there is the need to express fuzziness, possibilities, probabilities, penalties, costs, weights…etc. In general, problem specifications are difficult to be expressed by means of hard constraints only. While problems may become over-constrained easily, the need to avoid a result of no solution is acceptable in most cases. Many problems require finding best or optimal solutions with regard to one or even more optimization criteria agreed on or identified in the problem specification by relaxing some hard constraints. Others may occur due to vague attributes and, sometimes there is the need to include the uncertainty of a problem definition into final solution. All these problems need to apply some variety of preferences which have to be included into problem description and solution. That led us to conduct our research and investigate CSPs with preferences (also known as COPs) from both theoretical and practical points of view within this thesis.

1.2 Scope, Objectives and Proposed Solution

The thesis is concerned with educational timetabling problems in general. Usually, educational timetabling problems have various constraints that fit the requirements of the institution. These constraints depend entirely on the educational organization which may define them differently from other institutions. Emphasis

would be put upon the various constraints in terms of desires, requirements and conditions and this would result in a model that is difficult if not impossible to be solved as a generic timetabling model that is applicable to every institution.

The main goal of this thesis is to model a generic framework that solves a wide range of COP problems as well as timetabling problems, which will be our major focus. We introduce two generic solvers that solve two different timetabling problems using two completely different approaches. First, we present a modified backtracking with look-ahead algorithm with preferences to solve a basic teaching assignment problem. Second, in our main research in this thesis, we propose a multi-phase parallel hybrid metaheuristic solver to solve large examination timetabling problem.

The benchmarking datasets used as part of this research, introduced during ITC2007 (International Timetabling Competition 2007), are also discussed in brief. We present the results of our techniques in relation to both sequential and parallel approaches in comparison to the published results and provide a comparison between the outlined technique and those of the competition contestants. We also go over some of the other features and objectives of the developed solver.

Over the past 15 years, a number of approaches have been successfully developed to solve different types of educational timetabling problems with very good quality solutions. Typically, when a new methodology is proposed, its performance is measured against the best results found so far for a specific set of benchmark

datasets. Our intention is not only to get competitive results but also to show that the generic solvers and methodologies we are proposing can work with different types of problems once they are modelled in a correct and complete manner that will make the problem easier to solve.

We want to emphasise that the core, methodology and the architecture of the solver should not change when the solver tries to solve a different problem once the problem modelled according to the solver main elements (Variables, Domain Values and Constraints). This is clearly explained in our proposed solvers with the introduction of two main sections, problem dependent and problem independent portions. It is the implementation of the problem dependent that is different from one problem to another. The problem independent part must not change if another problem is introduced.

To summarize, this thesis will validate how existing frameworks, algorithms and methodologies can be extended to solve two of the common timetabling problems; teaching assignment problem and examination timetabling problem. The results are competitive in comparison to the published results so far for the benchmarking datasets.

In the following two tables, we present the main objectives and contributions of the thesis.

Table 1 shows the five main objectives of this thesis.

#	Objective/Goal
1	To reconstruct, rebuild and validate a C# based COP Solver that uses backtracking with look-ahead forward checking algorithm to be extended to use preferences to solve teaching assignment problem.
2	To define, describe and construct a local search based COP solver as a generic solver for any constraints satisfaction problem to be modelled and extended to use multi-phase parallel local search algorithms with hybrid metaheuristics to solve examination timetabling problem.
3	To verify experimentally both frameworks and by using a well-known benchmarking datasets with the second solver to produce reasonable and competitive results and using datasets from the computer science department with the first solver.
4	To explore the applicability of Microsoft technologies to real world scientific domains with a focus on timetabling applications.
5	The entire Ph.D. work was inspired by the idea of extending existing algorithms and methodologies to create and build a generic solver that can be pluggable and extensible for modelling and solving specific problems.

Table 1. Thesis Objective and Goals

Table 2 shows the main contributions of this thesis. It also cross references thesis objectives.

#	Contribution	Objective Cross Reference
1	An extension approach to the backtracking with look-ahead forward checking method that adopts weighted partial satisfaction of soft constraints that has been implemented to the development of an automated teaching assignment timetabling system. This approach is detailed in Chapter 3	**1,3**
2	A configurable, pluggable and easy to use local search solver. The solver is named *"Incremental Dynamic Search Solver"*, which is a generic local search solver that starts with feasible solution and then improves the quality of the constructed solution. IDS solver is used to model and solve the problem of Large Exam Timetabling. The solver is detailed in Chapter 4.	**2, 4, 5**
3	A parallel technique as a plug-in functionality using *"Message Passing Interface, MPI"* method to be used in the proposed IDS solver to solve any COP problem using more than one solver instance that can be hosted on the same machine or on different machines. The method is detailed in Chapter 4.	**2, 4**

4	A multi-phase parallel hybrid metaheuristic approach to solve exam timetabling problem. Details are in Chapter 6.	**2, 3**
5	A pre-processing stage in solving examination timetabling problem which involves two phases prior to commencing local search; problem collections ordering and unspecified constraints discovery technique that is used to discover all hard constraints that are not usually explicitly defined in the problem. The technique uses a full representational graph of the problem. The technique is described and detailed in Chapter 6.	**5**
6	A novel approach that modifies Tabu Search algorithm and uses a dictionary data structure mechanism to represent conflicts. In this approach, instead of recording recent moves, we record moves with an accumulated count of conflicts that caused. The idea behind conflict dictionary is to work as memory storage for previous clashes to avoid their potential reappearance in future. The approach is detailed in Chapter 6.	**5**
7	A modification to the Extended Great Deluge (EGD) method to reduce the amount of time wasted on local optimum. The proposed modified method is used to solve large examination problem and detailed in Chapter 6.	**5**

Table 2. Thesis Contributions Cross Referencing Thesis Objectives

1.3 Thesis Overview

This thesis is presented in eight chapters. Chapter 2 introduces great details timetabling algorithms, techniques and frameworks. This chapter serves as literature review chapter for this thesis.

Chapter 3, we go through the examination timetabling problem definition. We also describe in detail the different models, formulation and specification of the examination timetabling problem. We also study the different benchmarking datasets that are used by most researchers.

Chapter 4 describes a constraint programming system, with a website as front-end to demonstrate a suggested solution technique of the Teaching Assignment Problem.

Chapter 5 introduces Incremental Dynamic Search (IDS) Solver, a local-search based solver, we propose, for solving general finite constraint satisfaction and combinatorial optimization problems. The solver's ideal goal is to offer a pluggable approach to CSP models and to be used in the remainder chapters as an experimental platform for solving large exam timetabling problem. In

In Chapter 6, we go through in details and discuss our proposed approach to model the large examination problem as well as the search algorithm pre-process phase used. We study further the one methodology that we used in our approach;

metaheuristics and how we hybridize four of metaheuristics methods to perform our search.

Chapter 7 presents our investigation, experiments, findings and evaluation to the approach we developed to solve the large exam timetabling problem. We also introduce in detail the exam timetabling problem benchmarking dataset that was introduced in the international Timetabling Competition 2007 which was used as our case study dataset.

We conclude our thesis in chapter 8 with thesis summary and future research directions.

2. Timetabling Algorithms, Techniques and Frameworks

This chapter provides a detailed study to the timetabling problem and how it is solved using different algorithms, techniques and frameworks. The COP approach is used as a model for representing the problems we are tackling in this thesis.

2.1 Constraint Satisfaction Problem (CSP) and Constraint Optimization Problem (COP)

Before we formally define the CSP, let us introduce the important components it includes. In any CSP, a number of requirements are defined for a finite number of variables in the form of constraints. The set of possible values, i.e. domains, for each variable is finite. A constraint, basically, states which value tuples are allowed for a certain subset of all variables. A constraint can be given either explicitly, by enumerating the tuples allowed, or implicitly, e.g., by an algebraic expression.

Definition 2.1 (Constraint Network) A C*onstraint Network (CN)* is a triple *P* = (*V, D, C*), where

- $V = \{v_1, v_1, \dots, v_n\}$ is the set of variables called *domain variables*;
- $D = \{D_1, D_2, \dots, D_n\}$ is the set of *domains*.. Each domain is a finite set containing the possible values for a corresponding variable,
- $C = \{C_1, C_2, \dots, C_n\}$ is the set of *constraints* which restricts the values that variables can simultaneously be assigned .

A constraint C_i is a relation defined on a subset $\{v_1, v_1, \dots, v_n\}$ of all variables, that is $\left\{D_{i_1} \times D_{i_2} \times \dots \times D_{i_{k_i}}\right\} \supseteq C_i$. In other words, C is a finite set of constraints that restrict the values that the variables can simultaneously take.

The set of variables in constraint c will be denoted by V_c. If the set V_c has only one or two elements, it is called *unary* or *binary* constraint(s), respectively. If V_c has more than two elements, then it is called *non-binary constraint*. A CN is a *binary CN*, if all of its constraints are unary or binary. Binary CN play a special role, as any CN can be transformed into an equivalent binary CN by the so called *constraint binarization*. In practice, nonetheless, the transformation process may introduce too many additional variables with large domains which may significantly complicate finding the solution of a new problem [72].

Any CN can be represented by a *constraint graph*, where vertices represent CN variables which any two vertices are then connected by an edge if and only if there is a constraint referring to both variables [72].

Definition 2.2 (Assignment) Let $P = (V, D, C)$ be a CN, an assignment of the variables from V is $\eta \subseteq \{v/a | v \in V\ ; a \in D_v\}$ where $\forall v/a\ , w/b \in \eta, v = w\ \Rightarrow a = b$. An element v/a of η means that variable v has assigned value a.

An assignment defined on the set of domain variables V is *complete*, $iff\ |\eta| = |V|$ (i.e. all variables are assigned). Otherwise it is called *partial*. The set of all possible

complete assignments $\Theta_V = D_1 \times D_2 \times \ldots \times D_n$ is called *solution space*, and any solution should be searched within this space.

Definition 2.3 (Satisfied Constraint) Constraint c is *satisfied* in assignment Θ (noted $\Theta \vDash c$) if all of its variables have been assigned a value such that corresponding value tuple belongs to c. Otherwise, constraint is called *unsatisfied*.

For any given constraint $c_i(v_{i_1}, v_{i_2}, \ldots, v_{i_m})$, we note $\neg c_i(v_{i_1}, v_{i_2}, \ldots, v_{i_m})$ which is unsatisfied when c_i is satisfied.

Definition 2.4 (solution to a CN) *Solution of CN* $P = (V, D, C)$ is a consistent complete assignment σ, i.e., all constraints in the constraints set C are satisfied.

A partial assignment Θ is *consistent* if all the constraints referring only to variables assigned by Θ are satisfied. Partial Constraint Satisfaction is based on the idea of assigning as many variables as possible keeping the problem consistent (No constraints violation) and later on some constraints may be relaxed so that the partial assignment solution can be extended to include non-assigned variables [159].

Definition 2.5 (CSP) *Given CN* $P = (V, D, C)$ is a CSP consists of determining whether a solution exists, finding one or all solution to P.

A CSP with no solution is called *over-constrained* or *inconsistent* and a CSP with more than one solution is called *under-constrained*. A constraint is *relaxed* if there are

further element(s) added to the relation. If elements are removed from the relation, then the constraint is *tightened*.

As indicted above, in over-constrained problems [87], a consistent complete assignment that satisfies all constraints does not exist. As a result, new definitions of problem solution were introduced as alternatives like "Partial Constraint Satisfaction" [87].

In real life applications, when searching for a solution, a decision might be taken to use different types of *path exploration* in the solution space. The search process may look to find one solution, all solutions, or frequently some best solution regarding given criteria. Such solution is searched within the so called *optimization problem* or sometimes called *Constraint Optimization Problems (COPs)*.

COPs can also be defined as a regular CSP with weighted (or cost) constraints, and the objective is to find a solution, that maximize the weight (or minimize the cost) of satisfied constraints. The regular constraints are called *hard constraints*, while the cost or weighted functions are called *soft constraints*. This indicates that hard constraints are to be unavoidably satisfied, whereas soft constraints only express a preference of some solutions.

Definition 2.6 *Global Cost function F (also called Objective function or Criterion function)* is a function defined over CSP variables to convert from variables set *V* to an ordered variables set X.

Minimal values of X with regard to ordering $\leq x$ are considered to be more desirable in the exploration path.

As result, a COP is a CSP with an *objective function F*. Considering an optimization problem, an assignment θ is *preferred* to the assignment δ, if the value of the objective function under θ is less than the value under δ, i.e., $F\theta < F\delta$. An *optimal solution* is such a solution θ that none preferred solution to θ exist.

Usually, any global cost function is an expression which evaluates to a real number and whose value should be minimized. Maximization problem can be effortlessly transformed into an equivalent minimization problem by basically negating the value of objective function *F*.

2.2.CSP and COP Solving Techniques

Both CSPs and COPs are NP-Hard problems but naturally, COPs problems are harder to solve than CSPs. COPs are harder to solve because they take into consideration conflicting assignments on the basis that they might lead to a better solution and hence they spend more time and use complicated techniques and heuristics in exploring paths that may or may not lead to relatively better solutions.

The solving of CSP can be achieved by modelling the problem as a COP combined with the assignment of some positive cost to all constraints defined by the CSP. Solving this problem by using a COP solver would eventually return a solution (partial or complete). If the solution has a cost of zero, it is also a solution to the CSP, or else, the

CSP has no solution. Both CSP solvers and COP solvers can stop once they find a full solution that has no conflicts or a cost of zero in the COP problem. However, a CSP solver always maintains local consistency during the search to not to violate any constraint. On the other hand, a COP solver cannot stop at a violated constraint since it is quite possible that the current conflicting assignments are part of a solution which noticeably has a cost greater than zero. And that is why a COP solver may need to explore more of the search space looking for better solution with less cost.

Search algorithms for solving CSPs (Constraint Satisfaction Problems), and hence COPs, usually come down to one of two main families: local search algorithms and systematic algorithms. Both families have their advantages and disadvantages. Many researchers are designing hybrid approaches as they seem more promising since those advantages may be combined into a single approach.

2.2.1 Systematic Search Algorithms (Tree Search)

Generally, CSP solving uses the technique of systematically exploring all possible assignments of values to variables by means of a tree search algorithm. This is also known as backtracking search where at every node of the search tree the solver tries to reduce the remaining search space by removing values from the variable domains that will not subsidize to any solution. This process is called pruning the search tree, and the pruning techniques that are applied in CSP solving are known as constraint propagation. These techniques enforce some form of local consistency, mainly arc consistency, on the sub-problems represented by the nodes of the search

tree. In many problems, the time saved by the pruned search space considerably outweighs the time spent on constraint propagation [144].

Backtracking occurs when no more possible value can be assigned to the current variable, or what is known as a dead-end. The biggest problem of such tree search-based algorithms is that they typically suffer from early mistakes in the search. This is known as "Early-Mistake problem" where a wrong variable value can cause a whole sub-tree to be explored with no success.

In [16], [73], two of the well-recognized techniques are proposed to improve the performance of backtracking search:

1. **Look-back enhancements:** They utilize information about search which has already been accomplished. "Back-jumping", "Dynamic Backtracking" and "Learning" [148], [171], [187] are examples of look-back techniques, which evaluate the circumstances of failures during the search with the intention of determining healthier backtrack points or new constraints.
2. **Look-ahead enhancements:** They utilize information about the search space that has not been explored. **Filtering techniques** is one example of such techniques which allow the early pruning of sub-parts of the search space that would necessarily lead to a dead-end; as well as heuristics for **variable or value ordering** [160].

Look-back and look-ahead techniques can reduce the significance of the early-mistake problem, but unfortunately, they cannot solve it completely. They tend to require a whole sub-tree to be explored when a decision is needed to be undone if a proof of inconsistency has to be found.

2.2.2 Local Search Algorithms

Local search, on the other hand, carries out an incomplete exploration of the search space by repairing an infeasible complete assignment. Local search methodology operates in a manner that moves from one complete infeasible assignment to another, usually in a non-deterministic way, conducted by the help of heuristics. In most cases, the only disadvantage of local search algorithms is that they are incomplete. The reason for this is that they lack the guarantee to find a feasible complete solution that satisfies all hard constraints. Instead and in contrast to tree search algorithms, they do not suffer from the early mistake problem. Furthermore, local search algorithms are known to be more effective than systematic ones in finding a solution with regard to response time and quality.

2.2.3 Hybrid Solvers

In literature, the idea of the cooperation between local and systematic search have been studied by many researchers. Those hybrid approaches have led to good results on large scale problems. Three categories of hybrid approaches can be found in the literature [70], [145], [166], [204], [147]:

1. Performing a local search before or after a systematic search. This technique starts either start with a local search to get a partial solution that can be improved with a systematic search. This technique also can start with a systematic search until a dead-end is reached. At this point, it performs a local search phase which makes local changes on the current partial solution.
2. Performing a systematic search improved with a local search at some point of the search: at each leaf of the tree (i.e., over complete assignments) but also at nodes in the search tree (i.e., on partial assignments);
3. Performing an overall local search, and using systematic search, either to select a candidate neighbour or to prune the search space. Local search algorithms mainly will work upon a total instantiation of the variables where systematic-based search algorithms work upon a partial instantiation of the variables. In this method, the systematic algorithm only used to serve that specific purpose of selecting a neighbour or eliminating parts of the search space.

In chapter 6, we present a hybrid approach that falls optionally into the third category. Indeed, because the solver algorithm, we are proposing, is pluggable and extendable, it is the responsibility of the implementer to use the original systematic search neighbour selection or to provide another heuristics that is a problem specific as we will see in chapter 6. Our main objective is to show that the solver algorithm enhances the local search algorithms, and can significantly improve their behavior too. But first we need to go on some of the hybrid algorithms in literature.

2.2.3.1 Constrained Local Search Algorithm

The constrained local search algorithm presented in [147] adopts the approach of randomizing the backtracking part of a systematic search algorithm. This can be achieved by allowing backtracking to occur on random selected variables. It has an integer parameter called the noise level which states how many variables the algorithm can backtrack. The constrained local search algorithm iteratively extends a partial feasible assignment through assigning a selected value to a selected variable.

On selecting a value, only values consistent with the existing assignments are taken into account. The search continues until a complete assignment is found or it reaches a dead-end. When a dead-end is reached, which means that an unassigned variable with no value consistent with the existing assignment is selected, a given number of variables are unassigned, specified by the noise level parameter that the algorithm adopts, selected at random or using heuristics. A un-propagation process (backtracking) occurs and the algorithm then continues extending the partial assignment over again.

2.2.3.2 Decision Repair Algorithm

The decision repair algorithm, proposed in [109] is one of the algorithms that falls into the third category above. It can be seen, either as an extension of the classical depth first tree search algorithm with the introduction of a free choice of the variable to which to backtrack in case of inconsistency, or as a local search algorithm in the

space of the partial consistent variable assignments. It also can be seen as a hybridisation between local search and systematic search algorithms.

The algorithm repetitively extends a set of assignments which are called "*decisions*" in the algorithm, satisfying all the hard constraints, which is similar to systematic search algorithms. When a dead-end is reached, the algorithm implements a local search to repair these assignments. The dead-end is identified when the decisions become inconsistent. Once the inconsistent decisions are repaired, the construction of the solution continues to the next dead-end.

The algorithm starts with an empty solution or a partial solution which is a result of a set of decisions. It first applies a filtering technique. When no inconsistency is detected, the algorithm adds a decision that extends the current partial solution, and the search continues.

2.2.3.3 Composite Local Search Algorithms

One of the unique properties of the Local Search pattern is that different techniques can be combined, rotated and interchanged to produce novel or adapted complex algorithms.

Several approaches of combination have been proposed over the years. An example of composition of basic techniques is the WalkSAT algorithm for the satisfy-ability problem [174]. It alternates a Local Search phase with a random perturbation phase

called random walk that is used to escape the local optimum and to start again the search in a new region of the search space.

Iterative Local Search algorithms presented in [141], [102] combines local search with maintaining arc consistency and a conflict-based heuristics. The approach attempts to move from one partial or complete feasible (no violated constraints) solution to another (typically better) through repetitive assignment of a selected value to a selected variable. The search is terminated when the requested solution is found or when there is a timeout expressed.

2.2.4 Distributed and Parallel Constraint Solving

In contrast to composing a constraint solver from its fundamental parts, it is sometimes essential or desirable to distribute the solving processing. In this case, the solver is composed of a number of cooperating processes. One of the main reasons of doing so is that the CSP that needs to be solved is partially distributed, where it is difficult or undesirable to gather all constraints, and apply consistent, systematic and centralized solving methods. Such problems are generally denoted to as "*distributed constraint satisfaction problems*" or DCSP. Parallel CSP is another way in which distributed solving can be valuable.

2.2.4.1 Distributed Constraint Propagation

Distributed CSP algorithms have been researched in [137], [136]. The idea in the proposed algorithms is mainly concentrated around using constraint propagation

distribution or constraint propagation in parallel. The approaches involve some sort of communication during propagation which may affect the parallel performance that is intended in such approaches.

2.2.4.2 Distributed Search

Yokoo [203] has established numerous algorithms for distributed search, specifically for the DCSP problems. The fundamental hypothesis is that the DCSP variables are distributed among a set of agents in which these agents propose values for their variables to each other. Naturally, it is highly anticipated that such algorithms run asynchronously. They rely as little as possible on synchronization and external coordination, in order that the agents remain independent in the execution and decision taking of the search algorithms.

2.2.4.3 Parallel Search

The idea of parallelism in constraint solving is about managing parallel search by means of different instances of the solver, running on different processors explore different parts of the search tree in order to reduce the execution time. However, parallel processing suffers from a common issue which is achieving a good load balance. Load balance is about avoiding some processors become idle, while others will be busy doing most of the search. The reason for that is that, in most CSP and COP problems, the search trees can be irregular and unbalanced. To solve the issue, usually, a good solving algorithm exploits a dynamic load balancing pattern.

2.3 Timetabling Overview

Timetabling is defined as all activities in connection with making a timetable. According to Canadian Oxford Dictionary (2nd Edition) [48], a *timetable* is "*a plan of times at which events are scheduled to take place, especially towards a particular end".* The events are typically meetings between people at a specific location. Consequently, a timetable identifies which people meet at which location and at what time. A timetable must meet a number of requirements and should satisfy the desires of as many attendees as possible. The timing of events must be scheduled so that no-one can be in more than one event at the same time.

One of the first published works on timetabling was done in 1960's [7], [66]. More recent, since 1995, a large number of timetabling research has been presented in the series of international conferences on Practice and Theory of Automated Timetabling (PATAT). Papers on this research have been published in conference proceedings, [41], [40], [43], [38], [39]. Moreover, various researchers have been working on different areas of problems ranging from university timetabling ([12], [11], [34], [124], [163], [182], [179], [178], and [195]), school timetabling ([6], [60], [157], [156], [100], [168], [46], [112]), sports timetabling ([184]; [86]), and railway timetabling ([108], [49]).

Timetabling problems in general belong to the complexity class of "NP-Complete" [63]. Every so often, the word "timetable" is used interchangeably with other terms,

such as schedule, roster and sequence. Although all these terms are considered as synonymous, there are some profound dissimilarity between them [200], [192], [22], [14], [4] and [114].

The four terms are defined in [200] as follows:

Definition 2.7 (Timetabling): "*Timetabling is the allocation, subject to constraints, of given resources to objects being placed in space-time, in such a way as to satisfy a set of desirable objectives as much as possible"* [200].

Definition 2.8 (Scheduling): "*Scheduling is the allocation, subject to constraints, of resources to objects being placed in space-time, in such a way as to minimize the total cost of some set of the resources used"* [200].

Definition 2.9 (Sequencing): "*Sequencing is the construction, subject to constraints, of an order in which activities are to be carried out or objects are to be placed in some representation of a solution"* [200].

Definition 2.10 (Rostering): "*Rostering is the placing, subject to constraints, of resources into slots in a pattern. One may seek to minimize (or maximize) some objective, or simply to obtain a feasible allocation. Often the resources will rotate through a roster"* [200].

The abovementioned definitions cannot categorize problems into several separate sets as they are loose definitions. Some problems may fit more than one of these definitions. Therefore, a lot of researchers use these terms interchangeably.

Sequencing is basically the ordering of a set of activities, stating in which order an activity should follow another. A clear example of sequencing would be the ordering of tasks, to be processed, through machines in a factory. Sequencing may also take into account the costs related to one particular task following another.

Rostering is the assigning of resources into slots satisfying a number of constraints. A good example of this is the n-queen problem, where an N queens need to be placed in a $n \times n$ grid, in such a way that at no queen is in conflict with another (not same row, column, nor diagonal).

A timetable demonstrates at what time a specific event takes place; usually it includes the duration of that event. A train or bus timetable that displays trips which to be started on different routes and terminals. This type of timetable does not show passengers exactly which driver or vehicle is assigned to the trip route or if there is a change of drivers…etc. This kind of allocation of vehicles, drivers and other resources to routes is part of a scheduling process. Nevertheless, to guarantee compatibility between timetabling and scheduling problems, the problem is frequently assessed by researchers as one problem.

2.4 Educational Timetabling

Every semester and each year, educational institutions face the problem of scheduling courses, teaching assignments and examinations. As a result of the increasing number of students, courses, examinations, assignments, and complexity of resources and constraints (rooms, labs, and other constraints), the difficulty of the problem intensifies and the need for automated timetabling system that can generate realistic, high quality and feasible timetables is often vital. The timetabling process at educational institutes (i.e. schools and universities) is constructed using either completely manual timetabling; semi-automated timetabling or fully automated timetabling.

One of the earliest surveys is the one that was conducted during the year of 1988 by Comm and Mathaisel [61], involved 1,494 U.S. faculty registrars. The survey determined that most registrars felt that their manual or semi-automated systems are not efficient and there was enormous potential market for automated timetabling systems. The survey indicated some vital factors may affect a successful computerized system. These factors include the system should produce good quality timetables; it should be usable; it should allow some user involvement, wide-ranging and compatible with any existing systems.

Burke et al. [30] also conducted a survey in 1996 that involved 56 British Universities. The survey was concerned with three aspects: how registrars tackle their

examination timetabling problem. What they use to solve it, manual, semi-automated or automated system and what features they were considering for a good quality examination timetable. The survey concluded that only 58% (32 out of 56) of the universities specified that a computer is used at some stage to produce the final examination timetable and 21% (11 out of 56) of universities have an automated system for timetabling. Only 2 universities out of those 11 use commercial software while the other 9 used in-house developed systems.

Schaef A. in another survey [165] classifies educational timetabling problems into three main categories as follows:

1. **School Timetabling:** The scheduling of all classes of an elementary or a high school into any number of timeslots over one week period avoiding teacher and a group of students meeting at the same time. The classes should be pre-assigned a teacher and a group of students.
2. **Course Timetabling:** The assignment of a set of university courses to time periods and rooms over a week, minimizing overlaps of lectures of courses having common students and avoiding scheduling room or teacher to more than one course at the same time.
3. **Exam Timetabling:** The scheduling of the exams of a set of university courses to a finite number of rooms over a time period, such that all examinations are scheduled, avoiding overlap for exams having common students, and spreading the exams for the students as much as possible.

Although course scheduling in schools and universities may appear to be comparable, in fact there are some differences. On one hand, a school typically consist of similar class sizes for all of its courses. Furthermore, in school course scheduling usually the same group of students is attending the same set of courses. On the other hand university courses have more convenience in course registration as students have the choice to select which courses to join. Also, university course timetabling and examination timetabling also contrast in terms of specification and constraints involved. In course timetabling student can register, although is not preferable, in two overlapping courses whereas in exam timetabling students cannot attend more than one exam at the same time.

Carter and Laporte [52] classify academic timetabling, using other assumptions and consideration, into five categories:

1. **Teacher Assignment**: only considers the assignment teachers to already scheduled courses, it assumes the courses have been allocated to timeslots and classrooms.
2. **Class-Teacher Timetabling:** only reflects on the scheduling of the teachers' teaching times making sure that no teacher teaches more than one classes at the same time. It assumes that the teachers are already assigned to classes they teach.
3. **Course Scheduling**: addresses only the allocation of the courses to time and classrooms.

4. **Student scheduling**: focuses on the scheduling of courses to timeslots in a way that the courses taken by the same student are not scheduled simultaneously.
5. **Classroom assignment**: only allocates classrooms to courses considering resources needed for the courses, such as special equipment, seating capacity, time constraints…etc.

2.4.1 University Timetabling

University Timetabling Problem (UTP) can be defined as the task of assigning a number of events (i.e. lectures, exams, meetings…etc.) to a limited set of timeslots, rooms and other resources in compliance with a set of constraints. UTP is usually modelled as Constraints Optimization Problem (COP). The reason for that is that part of the problem's set of constraints can be categorized as soft constraints where preferences are identified with some violation penalties and the objective is to reduce the overall total cost. Corne, Ross and Fang [65] categorize timetabling constraints into five main classes:

Unary Constraints: They involve only one event, for instance the constraint "event *x* must not occur on a *Wednesday*" or the constraint "event *x* must be assigned to timeslot *y*"

Binary Constraints: they involve pairs of events. For example the constraint "event *x* must occur before event *y*". Another well-known binary constraint is the *event clash* constraint. This constraint states that if a specific resource is required to be present in a two or more events then these events must not be assigned to the

same timeslot. Also, chains of binary constraints are considered as binary constraints. A clear example of this "event *x* must occur before *y* which must occur before *z*".

Capacity Constraints: They are concerned with resources capacities (i.e. room capacities). For example "All events should be assigned to a room which has a sufficient capacity"

Event Spread Constraints: They are concerned with requirements such as the "spread out" or "tying together" of certain events within the timetable in order to ease the burden of students or teachers' assignments or to approve certain university's timetabling policies or regulations.

Agent Constraints: They are concerned with the preferences of the people who will use the timetables. An example would be the constraint "teacher *x* wishes to be assigned a particular event on Thursdays" or "professor *y* prefers to teach 3 days per week in the afternoon".

2.5 Time Tabling Algorithms and Methodologies

In literature, many researchers proposed different solutions for the timetabling problems for both schools and universities. These proposals were applied using different techniques that were developed to solve specific instances of the different timetabling problems. As a result of the different proposals implementations, a number of surveys on automated timetabling problems have been published that

classify timetabling problems and their solution algorithms. Most of these approaches started with modelling timetabling problems as COP with no to small number of non-complicated constraints. Later on as more research is contributed to this area and further advances have been reached, the problem started to be modelled as COP as more complicated constraints added to the problem.

One of the earliest surveys was one presented by Carter [51] in 1986 that provided a comprehensive overview of practical applications and strategies for solving the examination timetabling problem from 1964 to 1984. The applications were based on graph colouring heuristics designed to solve specific problems in particular schools. However, there was no comparison between the approaches used and consequently any newer examination timetabling application had slight to no foundation for choosing the best approach to start and work with.

Ten years later on, Carter and Laporte [52] used the previous survey as a basis for a new study that classified the algorithms into four categories: cluster methods, generalised search strategies, sequential methods, and constraint based approaches. Most algorithms, at that time, were designed to solve only the simple timetabling problems with no to small number of non-complicated constraints. Survey authors then started to publish a set of problems as standard problems to inspire more progressive research into timetabling problems. The first set was tested and described by Carter et al. [52].

Burke et al. in 1997 [22] outlined the necessity and significance of an automated timetabling system where an overview of some timetabling methods was presented. Schaerf A., in 1999, [165] published a survey on automated timetabling, classifying it into three main categories, using different point of view: school, course and examination. In that paper, a detailed overview of the different approaches to timetabling problems was presented. This includes basic search methods, optimization problems, and variants of the timetabling problem. It concluded with solution approaches published in the literature up to that year.

In 1991, Lotfi and Cerveny [123] proposed a multi-phase algorithm to solve a final exam scheduling problem. Van Hentenryck and and Saraswat in 1997 [186], and Cambazard et al. in 2004 [47] introduced an algorithm for solving course timetabling problem using constraint programming and constraints relaxation techniques to solve timetabling problems.

Burke and Petrovic et al. in 2002 [26] and in 2004 [29] presented a summary of university timetabling different problems and the latest approaches and methods in automated timetabling. The following section covers some of the common methods used in solving university timetabling with an emphasis on examination timetabling problem.

2.5.1 Cluster Methods

The method was first classified by Carter and Laporte [52] in 1996. It solves any university timetabling problem by dividing the problem into two steps. The main idea of the method is to split the events into groups or clusters, which satisfy hard constraints. These groups, then, are assigned to timeslots to satisfy soft constraints. In this approach, a solution may be found quickly but it may lead to a poor quality timetable if there are numerous dependencies between events of different clusters.

The two steps can be summarized as follows:

Step 1: Compatible events which have no or limited number of conflicts are grouped in a block or cluster.

Step 2: The cluster is arranged into specific periods trying to minimize a defined objective or satisfying specific constraints.

Both steps can be illustrated further with the consideration of examination timetabling problem.

Step 1 can be accomplished in different ways. In White and Chan paper [193], non-conflicted examinations are combined by ordering examinations by size. Leong and Yeong [121], examinations are arranged in descending order by the number of conflicts. Lotfi and Cerveney [122], examinations are ordered by the number of conflicted exams multiplied by the number of conflicted students with all other examinations.

The purpose of step 2 is to sequence the cluster while trying to minimize the number of students with consecutive examinations. White and Chan [193] approached the problem as a travelling salesman problem where the city is represented by a cluster and the cost to travel from city "A" to city "B" is denoted by the number of students who are required to sit examinations in both clusters, "A" and "B". Consequently, finding a minimum cost tour is equivalent to determining a sequence for a cluster that minimizes the number of students with two-in-a row examinations.

2.5.2 Sequential Methods

Sequential methods appeared in literature as early as 1964. Cole (1964) [59]; Broder (1964) [18]; Desroches et al. (1978) [74]; Laporte and Desroches (1984) [119] use sequential methods which are discussed in Carter's survey (1986) [51]. Sequential methods are usually known to be divided into two phases; a construction phase for initial solution and a successive improvement or enhancement phase. The construction phase involves a constructive heuristic that creates an initial solution. The algorithm should then selects events according to a sequencing strategy and assigns it to either the best feasible period to minimise the cost or to maximize the weight of its solution objectives or to the first feasible slot to minimise number of slots. The second phase is the improvement phase that involves alteration to the current solution by using modifications towards improvements. The improvements mechanism can be a process of selecting from different generated timetables from the

first phase. It can, also, involve performing slots or periods swaps on the lookout for improvement.

Constructive heuristics are used in the initialization phase of sequential methods and in constructing near optimal or good quality initial solutions. Constructive heuristic is typically an iterative algorithm that chooses an element from the problem's variable list and tries to assign to it a *feasible* value from its domain using a strategy that optimizes cost. Random techniques, greedy techniques and adaptive techniques are examples of constructive heuristics that can be used.

The main idea behind using heuristics in sequential methods is that it is far more efficient to find good quality or near optimal solutions than best solutions that turn out to be costly. Reeves [153], defines a heuristic as follows:

"A heuristic is a technique which seeks good or near optimal solutions at a reasonable computational cost without being able to guarantee either feasibility or optimality, or even in many cases to state how close to optimality a particular feasible solution is."

Timetabling problems were typically modeled, to be solved via sequential methods, as graph coloring problem. Graph vertices represent problem events. Conflicts between events are represented by graph edges between the vertices corresponding to the conflicting events. This graph may be colored in such a way that no adjacent vertices have same color where colors represent problem's time periods. As a result a feasible coloring of the graph confirms a conflict free timetable.

2.5.3 Metaheuristics

Metaheuristics techniques are considered to be one of the most successful techniques that have been applied to the COPs in the past 12 years or so. As of the time of the current research, metaheuristics can be divided into two sub-categories; local search-based and population-based metaheuristics.

Local search-based metaheuristics methods include tabu search [195] , [194], simulated annealing [182], [83], [181], [85], variable neighborhood search [45] , [135], [101], great deluge algorithms [24], [202], and greedy randomized adaptive search procedures (GRASP) [56]. These methods are distinguished from each other only by two main factors; their neighborhood structures and moving strategies. They usually start with a partial solution and involve searching in its neighborhood in an aim to find a complete feasible solution. Otherwise, if a complete feasible solution is used as a start, they try to improve its overall value.

Contrary to local search-based metaheuristics where a single solution is typically enhanced through an iterative process, population-based metaheuristics is based around multiple solutions. These methods includes genetic algorithms [163], [178], [179], [161], memetic algorithms [36] , [37]. Population-based metaheuristics compose some of the most successful techniques that have been applied to the exam timetabling problem in the last 10 years or so.

Generally speaking, using heuristics to solve COPs are problem-dependent. Methods that involve heuristics that work well for an instance of one problem may or may not be used to solve another instance of another problem. Therefore, considering strategy techniques that are more general, robust and can be applied to as many as possible of different problems is essential. Most of present local search algorithms are regarded as metaheuristics since they embrace some control, regulation or guidance over heuristics.

Glover and Laguna [94] define a metaheuristic as follows:

"A metaheuristic refers to a master strategy that guides and modifies other heuristics to produce solutions beyond those that are normally generated in a quest for local optimality. The heuristics guided by such a meta-strategy may be high level procedures or may embody nothing more than a description of available moves for transforming one solution into another, together with an associated evaluation rule."

Voß et al. [189] defines a metaheuristic as

"An iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high quality solutions. It may manipulate a complete (or incomplete) single solution or a collection of solutions at each iteration. The subordinate heuristics may be high (or low) level procedures, or a simple local search, or just a construction method."

Over the years, metaheuristics have been effectively applied to solve a comprehensive list of combinatorial optimization problems [153], [189], [142], [143], [1], [64], and [95]. In this list [188], simulated annealing, tabu search, evolutionary algorithms, GRASP and VNS have been applied. In the following sections we look at some of metaheuristics methods applied in literature.

2.5.3.1 Local Neighbourhood Search

Local search is an iterative algorithm that starts with an initial solution, searches the solution space by generating a sequence of solutions. Local Search algorithms are called single-based solution metaheuristics search algorithms. At each step; it selects, from the neighbourhood of the current solution, the next solution that it should move to.

To be able to solve a combinatorial optimization problem using local search, a list of assumptions have to be taken into account:

- *S* denotes a set of solutions and *s* denotes a single solution
- *f(s)* is function that calculates the cost of solution *s*
- An overall objective, for instance *maximize f(s) or minimize f(s)*
- A neighbourhood mapping function, *N(s)*, that maps a solution *s* in *S* to a set of related solutions in *S*.

Based on these assumptions, the general algorithm for local search is considered to be as follows:

```
initilize(s)
While not terminate(s) do
        s' = generate (s)
        if accept(s') then
            s = s'
        endif
endwhile
```

Figure 1 - Local Search Algorithm

The algorithm starts with an initial solution generated randomly. The *terminate(s)* function controls when the algorithm should stop searching. The *generate(s)* function generates a new solution selected from the neighbourhood. The *accept(s')* function determines whether the current solution s will be substituted with the newly generated solution s'. The functions; *terminate(s)*, *generate(s)* and *accept(s')* within the loop establish the improvement phase for the algorithm that determines which of the solutions in neighbourhood *N(s)* will substitute the current solution. The type of heuristic algorithm used in the improvement phase determines the performance of the whole local search algorithm.

Here we list some of these heuristics:

- **Random Walk:** It selects a solution randomly from the neighbourhood of the current solution and accepts without any condition. It is considered as the most simple and least valuable heuristic if it is used on its own. However, it is usually used in combination with other techniques to get better results.

- **Hill Climbing:** This is the most known local search technique. It is also known as *Simple Descent or Gradient Descent.* It works as follows; in any iteration, a candidate solution is selected randomly. If the candidate solution is better (whatever 'better' means here), it accepts the candidate solution and it replaces the current solution.
- **Steepest Descent:** It operates on all candidate solutions in the neighbourhood of the current solution. It tries to determine a better solution than the current solution. If it cannot find any, it terminates the search. If the neighbourhood is large, the technique can be slow. This may be avoided by limiting the number of candidate solutions list. Rules can be pre-set on how to limit the number of candidate list.

The main problem with all local neighbourhood search techniques is that they can easily get trapped in what is known as *local minima (or local maxima)*, where all neighbouring solutions are equal or worse than the current solution. Local search techniques cannot guarantee to find an optimal solution and do not necessarily search all the solution space. The performance of a local search is entirely dependent on several factors; here are some of them [157]:

- How the initial solution was generated.
- How the decision is taken regarding accepting a new solution.
- The size of the neighbourhood.

- The type of neighbourhood structure and whether it is single or multi-neighbourhood.
- How efficient the used neighbourhood search techniques which can affect enormously the performance of local search.

One solution to this problem is to conduct a ***multi-start local neighbourhood search***. This consists of a series of local searches, each one starting at a random point in the search space. The best result is saved from all the searches and is returned as the best solution. But, unless we look at all possible solutions we are still not guaranteed to find to global optimum.

Another type of problem we may find with local neighbourhood searches is finding a ***plateau.*** This is an area where the search space is flat so that all neighbours return the same evaluation. If a search hits this type of problem the usual solution is to conduct a random walk until you find an area that starts to give better quality solutions.

Most of these issues are addressed in the more recent metaheuristics techniques such as simulated annealing, tabu search, genetic algorithms, variable neighbourhood search…etc.

In educational timetabling problems and scheduling problem in general, Hill Climbing is now commonly used in multi-phase solvers with other metaheuristics [30], [36],

[167], [128] and [111]. Hill Climbing can also be used as a standard algorithm where new algorithms' performance can be compared against [65] and [31].

2.5.3.2 Simulated Annealing (SA) Algorithm

Simulated annealing (SA) is an extension to Hill Climbing algorithm. It differs from HC in that it implements a more tolerated acceptance rule by accepting moves of poorer quality if they conform to some probabilistic approaches. Inherently, it still accepts improving moves.

Simulated Annealing (SA) is motivated by an analogy to annealing in solids. The origin of SA algorithm goes back to 1953, where the method is based on ideas motivated by the Metropolis algorithm for statistical mechanics by Metropolis et al. [129]. The algorithm in this paper simulated the cooling of material in a heat bath. This is a process known as annealing. If a solid is heated past melting point and then cooled, the structural properties of the solid depend on the rate of cooling. If the liquid is cooled, slowly enough, large crystals will be formed. However, if the liquid is cooled quickly, the crystals will contain imperfections. Metropolis's algorithm simulated the material as a system of particles. The algorithm simulates the cooling process by steadily lowering the temperature of the system until it converges to a stable state.

In 1982, Kirkpatrick et al [113] took the idea of the Metropolis algorithm and applied it to optimization problems. The idea is to use simulated annealing to search for feasible solutions and converge to an optimal solution. In simulated annealing, the

temperature and the change in the evaluation function determine how likely it is that a worsening move will be accepted. A high temperature gives a high probability of acceptance and a low temperature causes moves to be rejected with a high probability.

- **Acceptance Criteria**

The law of thermodynamics states [2] that at temperature, *t*, the probability of an increase in energy of magnitude, δ_E, is given by

$$P(\delta_E) = \exp(-\frac{\delta_E}{kt})$$

Where *k* is a constant known as Boltzmann's constant.

The simulation in the Metropolis algorithm calculates the new energy of the system. If the energy has increased then the new state is accepted using the probability returned by the above formula. If the energy has decreased, then the system moves to this state. The SA is executed for a certain number of iterations at each temperature and then the temperature is decreased. This is repeated up until the system freezes into a stable state.

Simulated annealing has been implemented successfully to solve numerous optimization problems. Dowsland [83] presented a comprehensive study to SA techniques and references in literature. Abramson et al. [5] have successfully applied simulated annealing to school timetabling problems. Thompson and Dowsland [181]

solved examination timetabling problem in two phases. The first phase produced a feasible timetable, which eliminated clashes and satisfied resources constraints. The second phase removed all infeasible solutions from the solution space and searched the remainder for a solution that optimized the secondary objectives. Their experiments show that the solution quality depends not only on the cooling schedule and neighbourhood but also on the way it is sampled. Although it is one of the most successful metaheuristics methods applied to timetabling problems, criticisms against it have been made because of the requirement to find a cooling schedule for a particular instance of a certain problem.

2.5.3.3 The Great Deluge (GD) Algorithm

The great deluge algorithm (GD) introduced, by Dueck [84], in 1993. It can be used as an alternative to simulated annealing where the algorithm begins with a "water level" equivalent to its initial solution in SA, and a pre-set rate to decrease water level in each iteration. This predetermined rate is the only parameter for this algorithm. The algorithm accepts worse solutions if the penalty cost is less than the "water level". Dueck applied this algorithm to the travelling salesman problem. The level was decreased by at least 0.01 or by the difference between the level and the length of the current tour divided by 500.

As a result of the benefit of using less parameter, the great deluge algorithm has been used satisfactorily in several other implementations of metaheuristics. Burke and

Newall [25] implemented it to an examination timetabling problem and produced adequate results on 13 benchmark datasets [54]. In this research paper, the pre-determined rate to decrease the water level was called “decay rate”. The great deluge algorithm was found to perform better when compared to simulated annealing and hill climbing. It also performs better in most cases compared to other methods such as tabu search method, Di Gaspero and Schaerf [78], and graph colouring heuristics, Carter et al. [54]. Burke et al. [28] implemented a time-predefined simulated annealing and time-predefined great deluge algorithm.

The modified great deluge algorithm takes in two parameters instead of the original one. They are “time to run the algorithm” and “an approximation of the anticipated solution quality”. The decay rate is computed as the difference between the initial solution quality and the desired solution quality divided by time or number of moves in the desired time. It also enhances the acceptance criteria of great deluge algorithm by accepting improving moves even if they are greater than the level. The modified time-predefined great deluge algorithm was found to be better compared to other methods.

2.5.3.4 Tabu Search (TS)

Tabu search (TS) was proposed by Glover [93] in 1986 to solve originally combinatorial optimization problems. Based on that proposal, TS-based methods

have been known to be effective in several areas of optimization. Glover and Laguna [94] defined TS as follows:

"Tabu search is a metaheuristic that guides a local heuristic search procedure to explore the solution space beyond local optimality."

The basic idea of tabu search is to be an extension of steepest descent algorithm through combining adaptive memory and reactive exploration. Adaptive memory is used to memorize two vital elements of any problem; keeping track of the current best solution and storing information related to the exploration process. The algorithm, basically, starts with an initial solution S_0, then it iterates by exploring a subset $N'(s)$ of the neighbourhood, $N(s)$, of the current solution S. The variable with the lowest value becomes the current solution regardless of whether the value of the new solution is better or worse than the current solution.

By accepting a non-improving move will allow the search to escape local optima and continue to discover other search areas. However this will typically lead to cycling, which is, repeatedly moving between some set of solutions. To avoid this, it uses memory allocation to store a list of tabu moves. This list contains prohibited moves that satisfy some tabu restriction criteria for a predetermined number of iterations which is called "tabu tenure". Prohibited moves that are included in the tabu list will carry a status of *tabu-active*.

To make a solution a tabu free, an aspiration criterion is used if the solution evaluation is of satisfactory quality and can prevent cycling. Although, this is useful for guiding the search to reach more unexplored areas and to produce good results, an appropriate choice of aspiration criteria is important. The aspiration criteria can be time-independent, time-dependent or aspiration by default. In Reeves's research paper [153], other aspiration criteria are discussed. They are aspiration by influence, aspiration by objective and aspiration by search direction.

Tabu Search uses a stopping condition to terminate the search. This could be a determined number of iterations, or it exceeds a specific number of iterations since the last improvement. The length of tabu list, the aspiration function and the cardinality of the set of neighbour solutions tested at each iteration are also examples of important parameters that can be considered.

Two important strategies used in tabu search are presented by Glover and Laguna [94]. They are intensification and diversification. Intensification strategies implicate changing variable and value selection rules to intensify the search to examine neighbours of best solutions. The main idea behind this strategy is that if specific areas contained good solutions in the past they may possibly produce better solutions in the future. The diversification strategy encourages the search process to explore unvisited areas and to generate solutions that may be considered to be significantly different. In this research, at least three approaches for defining the tabu list size;

fixed to a predetermined value, randomly chosen from a particular range or dynamically changed by modifying the value.

Tabu search was used to solve educational timetabling problems as early as 1991. Hertz [104] proposes a tabu search based solution to solve school timetabling with soft constraints. Schaerf and Schaerf [169] implemented tabu search techniques in scheduling classes to periods in a large high school. They represented their timetable as a matrix of integers such that each row represents a weekly assignment for a specific teacher. They used two types of moves which are single moving of a class to another period and double moves which are moving made by a pair of single moves. The algorithm used a tabu search with single moves included with a randomised non-ascendant method (RNA) using double moves.

The RNA is used to generate the initial solution and is applied again after tabu search reached non-improving state for a specific number of iterations. The cycle is repeated allowing tabu search to start in a different direction. The tabu list is of variable size so each move is added into the tabu list with a number of iterations selected at random within a predetermined range. As a result, the “tabu tenure” contrasts for each move. Each time a move is inserted the value of all the other moves in the list will be decreased and when it reaches zero the move is removed. The algorithm uses the simplest aspiration criteria of accepting a tabu move only if it improves the current best solution. The technique provided good results for schools of various

types. The timetable produced is better than the manual timetable and it was able to schedule 90-95% of classes.

Di Gaspero [76] introduced an algorithm that utilizes a multi neighbourhood strategy applied to examination timetabling. He applied a combination of tabu search with different neighbourhoods. He categorised these combinations into local search that specialized in optimizing the objective function, perturbing current solution or obtaining more improvement which he called recolor, shake and kick algorithms. These algorithms were applied in sequence until no further improvement and the algorithm ended with the kick algorithm. He used seven benchmark datasets and his results were better compared to the basic tabu search with a single neighbourhood.

There are other tabu search based algorithms in the literature which are different flavours of the original tabu search algorithm. We only list their references here [88], [78], [82], [194].

2.5.3.5 Genetic Algorithms (GAs)

Evolutionary algorithms are called population-based metaheuristics search algorithms. These include: genetic algorithms (GA) [105], Ant Colonies (AC) [81], Genetic Programming [117], Evolution Strategies (ES) [152], and Scatter Search (SS) [96]. Genetic Algorithms (GA) are evolutionary algorithms that are based on directed random search techniques presented in 1975 by Holland [105]. Goldberg [98],

Michalewicz [131] and Mitchell [134] did comprehensive review on genetic algorithms and their different applications.

Genetic algorithms basically iterate on a population of solutions represented by some encoding. Each member of the population consists of a number of genes (i.e. a unit of information). The algorithm begins by creating an initial population, usually randomly generated. Then, the solution members are evaluated by computing their fitness. Next, the selection procedure reproduces more copies of individuals with higher fitness values. The selection procedure influences the search direction towards promising search regions. Crossover (combining promising areas of two parents) and mutation (small random changes) are called genetic operators which are used to create new populations.

There are also other essential control parameters that involve in the genetic algorithm include the population size, crossover rate and mutation rate. A large population size escalates computational time; the crossover rate determines the frequency of crossover operation. A little care needs to be taken when selecting a mutation rate. If a mutation rate is too high, it may cause diversity in the population and might cause instability in the search process.

Ross et al. [161] applied GA to solve university lecture timetabling problem where each chromosome is a timetable solution enclosing a list of events. The algorithm uses

a population size of 50, gene-to-gene mutation, uniform crossover, tournament selection and elitist reproduction.

Burke et al. [37] used solutions produced by several graph colouring heuristics to initialize solution population of a genetic algorithm. A random element is used in solution initialization by using bias selection and tournament selection.

2.5.3.6 Other methods

There are other approaches mentioned in the literature. We mention, in brief, some of them that are related to solving timetabling problems. Greedy Randomized Adaptive Search Procedure (GRASP) is a multi-start iterative procedure. Each of GRASP iterations consists of a construction phase and an improvement phase [155]. Casey and Thompson [56] implemented an enhanced GRASP algorithm on examination timetabling problems. They solved a basic examination timetabling problem of scheduling examinations into timeslots so that conflicting examinations are assigned in different timeslots. The objective is to minimize the proximity of examinations in students' timetables.

Memetic Algorithm (MA) which was applied by Moscato [139], [138] is an evolutionary algorithm that includes knowledge of the problem in the form of approximate algorithms, heuristics, specialized recombination operators, local search techniques, etc. It is in some way similar to GA. The difference between memes and genes is that memes are ideas or concepts that are passed around and adapted to

the environment while genes are passed down without alteration. Most memetic algorithms use genetic algorithm and local search on each population member after each generation. Burke and Landa Silva [23] presented a review on strategies used by scholars using memetic algorithms for scheduling and timetabling problems.

Case-Based Reasoning (CBR) [116] is a knowledge–based pattern that attempts to use previous knowledge stored as cases to build solutions for current problems. The main idea in this methodology is not to start from scratch by reusing similar solutions that are already stored in a case base, but to adapt a retrieved solution to the new problem and subsequently enrich the case base with new knowledge or cases. Burke et al. [35], [21] and Qu [149] applied CBR to university course timetabling.

2.6 Conclusion

In this chapter the problem of educational timetabling problem was introduced emphasizing out the significance of examination timetabling and teaching assignment problem. Various algorithms, heuristics and metaheuristics techniques have been applied to timetabling and scheduling problems. It seems that there is a trend to use hybrid techniques [8], [90], [158], [175], [109], [147], in an effort to solve optimization problems. It is, also, notable that most of the algorithms, techniques and frameworks introduced in this section have one common feature which is an attempt to bring together new advances such that performance can be improved.

3. Exam Timetabling: Models, Formulation and Current Research Status

In this chapter we go through the examination timetabling problem definition. We also describe in detail the different models, formulation and specification of the examination timetabling problem.

We also present a theoretical investigation to the exam timetabling problem benchmarking in general. We examine, in much details, ITC 2007 exam timetabling problem, its different soft and hard constraints, variables and resources. This chapter also presents comprehensive information about ITC 2007 exam timetabling benchmarking datasets. We describe an overview of how the different problem datasets compare to each other regarding hardness and conflicts density. We also present the reason for the choice of these benchmarking datasets on which our new approaches and techniques are to be tested.

3.1 Introduction

When it comes to timetabling paradigm, university timetabling problem was one of the leading problems that encouraged researchers to study generic timetabling problems [59]. Recently, especially in large academic institutions, constructing timetables manually became more complex, tedious task and monotonous process as a result of introducing more modest, multiple and complex constraints to the

problem. This was indeed an outcome of the continuous increase of flexibility to syndicate different modules of the problem into one large problem.

Finding a good quality final time table is crucial to all parties that are involved in the full academic process; students, teachers, technicians, administrators, resources…etc. Poor quality timetable would be beneficial to some, while others might suffer. For instance, the figures of students, who pass exams, depend on many factors. One of these essential factors will be how these exams are spread out over timeslots adequately and evenly among all students.

University timetabling in general can be classified into two main groups [165]; course and examination timetabling. University course timetabling, in principle, can be divided into two subgroups; post-enrolment (PostEnroll CTT) and curriculum-based (Curriculum CTT) course timetabling. The difference lies in the fact that PostEnroll CTT timetables is constructed after the students have already selected their courses whereas in the curriculum-based case, the students have to take a set of courses that belong to a certain curriculum and hence timetables are all have been scheduled.

The exam timetabling problem (Exam TT), on the other hand, is an extensively studied combinatory optimization problem that is regularly faced by most universities. In recent years, the problem has been getting increasingly challenging as the number of students enrolling in universities increases. These students are enrolling into a broader selection of courses which include even growing number of

joint degree between these courses [128] which affect enormously the complexity of the problem.

3.1.1 Problem Definition

The basic Exam Timetabling problem is the allocation of a set of exams to a number of timeslots and rooms under the condition of satisfying a set of constraints. Different universities assess the quality of exam timetabling based on different views. This diversity has resulted in many different formulations of the problem in consideration of different sets of constraints ([52], [165], [30], [151]).

One of the known conventions used in any timetabling problem is to group the considered constraints related to the problem at hand into two main categories namely the hard constraints and soft constraints. This grouping aspect is what differentiates timetabling problems from many other types of combinatorial optimization problems. Hard constraints have a greater significance than soft ones and have to be satisfied. That is why timetables will only be considered as feasible if all of the compulsory hard constraints are satisfied. Soft constraints, however, are preferable to be satisfied if possible. Typically, the higher the number of satisfied soft constraints, the better quality the solution is. This, nevertheless, depends on the university's timetabling policies in addition to the satisfaction of the individuals who will use it. In section 3.4 we shall introduce both

hard and soft constraints in more details when we formulate the problem mathematically.

3.2 Problem Models

There are several modelling methods for exam timetabling problem. In this section we only describe two of them which are considered to be the most important ones. These two models are streamlined for the purpose of simplicity by taking into account only the critical constraints while disregarding other less important constraints. The important constraints are those that are common in many universities such as avoiding exam conflicts (also known as first order conflicts) and scheduling all exams.

3.2.1 Graph Model

Graph theory has played a significant role in the research on timetabling problems [23]. The graph colouring problem, specifically, has the objective of discovering the minimum number of colours that can be applied on nodes or vertices of a graph such that no two adjacent vertices have the same colour. Welsh and Powell [191] argued that there is a close relationship between graph colouring and timetabling. The graph colouring problem is known as NP-complete [110] because there is no efficient polynomial-time algorithm that can find the minimum number of colours (also known as chromatic number) for the graph.

The timetabling problem can be modelled as a graph model by scheduling all examinations into timeslots such that no two conflicting examinations (first order conflict) are scheduled in the same timeslots. Let us consider an exam timetabling problem with n examinations, $e_1, e_2, \dots, e_n$ and the goal is to schedule these exams into a predetermined number of timeslots. This problem can be symbolized as a graph, consisting of a set of n vertices where each vertex represents an exam e_i. An edge connects two vertices i and j, only if there is a conflict (constraint) between exams i and j. The edges can be labelled with value that represents a weight or cost to mark the actual number of students involved in that conflict. Also, the vertex can be labelled with a value that represents the number of students enrolled for the corresponding exam. The total number of edges from a vertex i is known as the *vertex degree* to specify the total number of exams in conflict with exam i. Obviously, with this information in mind, it can be considered as a graph colouring problem where each vertex of the graph is required to be coloured such that no two adjacent vertices connected by an edge have the same colour. Every unique colour relates to a timeslot and therefore two adjacent vertices or exams will not be assigned to the same colour or timeslot.

A solution exists if we can solve the problem by using $\boldsymbol{c}$ colours with the condition that $\boldsymbol{c}$ is less than or equal to the number of timeslots. The graph colouring heuristics also can be adapted to solve examination timetabling problems and they are still being implemented in recent approaches of metaheuristics.

3.2.2 Mathematical Model

Exam timetabling problem is a combinatorial optimisation problem that can be represented using a mathematical model. According to Williams [198] any mathematical model should conform to the following rules:

- Decision variables when chosen have to be related to manageable input. In other words, values that are given to variables should result in a solution that can be implemented without vagueness. In examination timetabling, the objective is find the most suitable timeslot for an examination. Consequently, variables must be related to all the examinations and for each variable, the value assigned is chosen from the slots available.
- Any problem must have an objective function such that an evaluation can be done for different decision results. This is because a value of any objective function should determine the quality of a solution. If it is a maximization objective a higher value is better than a lower value and vice versa for a minimization objective.
- Problem constraints that represent restriction or limitation on values of decision variable must be defined and determined without any ambiguity.

Based on these rules, a problem formulation is discussed in the following section. The mathematical model shall be discussed more in section 3.5 where we discuss current research state.

3.3 Problem Formulation

There are many formulations to the problem that were developed among the years by researchers. However, in this section we try to summarize the major concepts around formulating the problem in literature using two main formulations provided by Burke et al. [38] and Weare [190].

General examination timetabling formulation can be summarized as follows:

- $\boldsymbol{E}$ represents a set of $\boldsymbol{m}$ examinations $\boldsymbol{e_1}, \boldsymbol{e_2}, \dots, \boldsymbol{e_m}$
- $\boldsymbol{m}$ or $|\boldsymbol{E}|$ represents the number of examinations
- $\boldsymbol{L}$ represents a set of $\boldsymbol{n}$ *time*slots $\boldsymbol{l_1}, \boldsymbol{l_2}, \dots, \boldsymbol{l_n}$
- $\boldsymbol{n}$ or $|\boldsymbol{L}|$ represents the number of slots
- Each timeslot has a seating capacity $\boldsymbol{S}$
- $\boldsymbol{r_{ij}}$ is the number of students registered for exams $\boldsymbol{e_i}$ and $\boldsymbol{e_j}$
- $\boldsymbol{s_i}$ is the number of students registered for exam $\boldsymbol{e_i}$
- A final examination timetable (Final Solution) is represented by $\boldsymbol{T_{mn}}$ such that: $\boldsymbol{T_{ik}} = \boldsymbol{1}$ if examination $\boldsymbol{i}$ is scheduled in timeslot $\boldsymbol{k}$, otherwise $\boldsymbol{0}$.
- A conflict matrix represented by $\boldsymbol{C_{mm}}$ such that the total number of students who are sitting for both exams *i* and *j* is represented by $\boldsymbol{C_{ij}}$.
- An exam is given a penalty (or cost) if exam $\boldsymbol{i}$ is scheduled in slot $\boldsymbol{k}$. This penalty is represented by $\boldsymbol{P_{ik}}$.

Hard constraints are supposed not to be violated. Hence, a timetable that violates a hard constraint will be considered as *infeasible*. Infeasible timetables, in some special cases, may be inevitable and the concerned institute usually take exceptional measures to resolve the problem.

In section 3.1, during the introduction of exam timetabling definition, we went through exam timetabling problem hard and soft constraints. We shall now introduce them mathematically.

Talking specifically about exam timetabling problem, among all of their different formulation, there are three main constraints that are considered as common to all exam timetabling problems ([36], [57]):

- **Student Constraint**: Not a single student is taking more than one exam at any one time. Violation of this constraint is usually known as a "conflict".

$$\sum_{i=1}^{|E|}\sum_{j=1}^{|E|}\sum_{k=1}^{|L|} T_{ik}\, T_{jk}\, C_{ij} = 0$$

- **Capacity Constraint**: The total number of students taking an exam must not exceed the room(s) capacity where the exam is taking place. Correspondingly, for each period, there must be enough room(s) capacity for all the exams that are scheduled for that period.

$$\sum_{i=1}^{|E|} a_{il}\, s_i \leq S, \qquad \forall\ l \in L$$

- **Exam constraint**: This constraint refers to the fact that all examinations must be scheduled and each one of them ($i.e.e_1, e_2, e_3, \dots, e_m$) must be scheduled once and only once. If exam i and exam j are scheduled in slot k, there must not be any shared students in both exams. In other words, the number of students sitting for both exams i and j *which represented by* C_{ij} must be equal to zero, and this should be accurate for all exams already scheduled.

$$\sum_{k=1}^{|L|} T_{ik} = 1 \; \forall \; i \in m$$

Because of the impracticality of violating these three constraints, they are known as hard constraints that must be satisfied for any timetable to be considered as feasible. In contrast, there is another type of constraints which is known as soft constraints where it is desirable to maximize the number of satisfied ones. If the exam timetable final solution is considered only feasible if no hard constraint was violated, its quality is determined based on the penalty given if some soft constraints were violated. We determine the quality of an examination timetable solution based on the penalty given if certain soft constraints are violated. These constraints ([30]) include but not limited to:

- No student should take more than one exam in successive periods.
- No student should take more than one exam in one day.

- Some exams can only take place according to some constraints like preferred particular rooms, resources and/or periods only and not others.
- All exams should be scheduled in less than a specific number of periods or should be scheduled in a specific number of rooms.
- Large exams should take place in the first few periods in the examination's timetable to allow enough time for instructors to mark exams sheets.

These first two soft constraints can be seen as the most important soft constraint to satisfy as they affect students' potential success directly and can be summarized as:

- For all students, study periods between exams are appropriately sufficient and even.

It is also known as the "***proximity constraint***" and a "***proximity cost***" is specified when the proximity constraint is violated. Typically, a weighted proximity cost $\boldsymbol{x_s}$ is given whenever a student has to sit for two examinations scheduled $\boldsymbol{s}$ periods apart.

$\boldsymbol{P_{ik}}$, the total proximity cost if examination $\boldsymbol{i}$ is scheduled in slot $\boldsymbol{k}$, is as follows:

$$P_{ik} = \sum_{j=1}^{|E|} x_{|k-l|}\, C_{ij}$$

Where, $\boldsymbol{j}$ *is* an exam which is in conflict with another exam $\boldsymbol{i}$, and is scheduled in $\boldsymbol{l}$.

The objective is to minimize the number of conflicts between exams

$$\sum_{i=1}^{|E|-1} \sum_{j=i+1}^{|E|} \sum_{t=1}^{|L|-1} T_{it}\, T_{j(t+1)}\, r_{ij}$$

and to minimize the number of periods $|\boldsymbol{S}|$

And to minimize the total of proximity cost

$$\sum_{i=1}^{|E|} \sum_{k=l}^{|L|} T_{ik}\, P_{ik}$$

If the timetable has unscheduled exam, it is common to penalize each unscheduled exam with a relatively high number like a number with a three digit (thousands) that should be added to the total cost and reflects the quality of the outcome timetable. The proximity cost and weights were first introduced in [52] to measure and evaluate the solution quality and this is also used in this thesis following the rules of the benchmarking datasets used in our experiments.

3.4 Problem's Current Research State

The exam timetabling problem is a problem that occurs every year if not every term in universities. It is extensively studied by many operational research and artificial intelligence researchers as a result of its complexity and practicality. In the last 50 years or so, a comprehensive range of methodologies for solving the problem have been proposed and discussed in literature. These approaches can be divided into seven general categories [51], [26] and [150], graph-based sequential techniques,

clustering-based techniques, constraint-based techniques, metaheuristics techniques, multi-criteria techniques, hyper-heuristics techniques, and case-based reasoning techniques. In this section, we will go through every technique in more details.

3.4.1 Graph-Based Sequential Techniques

Generally speaking, any timetabling problem, without embedded soft constraint, can be easily modeled as graph colouring problem [51] [28]. Using this model, graph vertices represent problem exams (variables) and graph edges between vertices represent conflicts between exams (hard constraints) [192], [53], [33]. The objective is to colour the vertices so that there are no two adjacent vertices with the same color noticing that each colour represents a period in the timetable. To enhance the graph colouring model, numerous graph coloring heuristics have been proposed in the literature [18], [54], [199].

These heuristics are based essentially around ordering exams, (e.g., exams with the largest conflict first), and then each exam is assigned to a period in that order. These heuristics are not usually used on their own but rather hybridized with other search methods for efficiency reasons [40], [54], [78], [37], [27], [50], [9].

The main drawback of using them alone is their limitation on early assignments that may lead to the unavailability of feasible periods for exams left later in the construction process which leads to not getting to feasible solutions.

3.4.2 Clustering-Based Techniques

Like the name suggests, clustering-based techniques divide exams into groups (clusters) where exams within each group must satisfy all hard constraints. The groups are then assigned to periods with a secondary objective of minimizing soft constraints violations [193], [122], [13].

3.4.3 Constraint-Based Techniques

Similar to constraint logic programming [103] and constraint satisfaction techniques [17], exams are represented as variables with finite domains. Periods, on the other hand, are represented by the values within the variable's domain representing the exam. Any exam should be assigned to a period with the condition of not violating any constraint Values (periods). Later in the assignment process (search), when no value can be assigned to a particular variable, a backtracking technique should be adopted to enable the reassignment of values until a feasible timetable is completely constructed.

Like graph-based sequential techniques, constraint-based techniques are rarely used on their own since they usually cannot provide high quality solutions [17]. They are usually used in hybrid algorithms to find an initial feasible solution whose quality is then optimized by other exhaustive search methods [128], [69], [85].

3.4.4 Metaheuristics Techniques

Metaheuristics techniques are considered as the most successful techniques that have been developed to solve exam timetabling problem. The technique starts with constructing an initial solution. The initial solution can be a feasible or infeasible. After constructing an initial solution, a pre-determined heuristics based on problem domain (Exam timetabling problem in this case) is used to reduce cost and optimize solution quality. The techniques typically use a fixed set of heuristics, and are usually tailor made to solve a particular problem or instance.

Metaheuristics techniques have one downside which is they rely on parameter tuning and do not work consistently across different exam timetabling problem instances. This problem is motivated by the fact that metaheuristics are dependent on domain knowledge which ironically, their strength. In chapter 6, we will describe this technique in more details when we introduce our technique to solve exam time tabling problem using metaheuristics approach.

3.4.5 Multi-Criteria Techniques

Multi-criteria or multi-objective techniques are another group of exam timetabling solvers. Any real-world exam timetabling problem is usually defined with a number of soft constraints which identify more objectives of the problem. Most current approaches combine all objectives of the problem into one aggregating function. Multi-criteria optimization presents a more general and flexible approach by considering a collection of objectives, which enables all the objectives to be

optimized simultaneously. Besides, it identifies a better assessment and understanding of the problem by learning more on the relationship between the different objectives. These diverse objectives are usually conflicting in nature since they are considered from different viewpoints by the different parties that are involved in the timetabling process [52].

3.4.6 Hyper-Heuristics Techniques

Hyper-heuristics represent a completely different approach, contrary to other approaches, to exam timetabling. Rather than working on a search space of solutions, hyper-heuristics work in a search space of heuristics to select the best set of heuristics for solving the existing instance of the problem. The literature is full of this group of exam timetabling solvers [32], [178], [111], [45], [150], [9]. Most of them are motivated by the limitation of metaheuristics mentioned above and are designed at achieving a higher level of generalization which arguably most of the other techniques lack.

3.4.7 Case-Based Reasoning Techniques

Case-based reasoning techniques are relatively new approach which was encouraged by the study of human learning process where historical experience with results of specific problem is used to solve newer instances of similar problem. Regarding exam timetabling, the solutions of previously solved instances are

employed to assist the search of solutions to new problem instances. This technique has been used by [35], [149], [44] for exam timetabling.

3.5 Problem Benchmarking

Over the years, different competition instances as well as benchmarking problems were made available to researchers by different institutions [126]. The benchmark problems have been originated from several universities like the University of Toronto, Canada; University of Nottingham, England; Queens University, Northern Ireland…etc. [151].

Generally, the literature on examination timetabling can be categorized into two main groups:

- **Real life Problems.** This group focuses on solving examination timetabling problems from real life model usually faced by the institution where the researcher is located or introduced by some other researchers. Pairing in mind that there is no guarantee that anyone can validate the accuracy of the achieved results other than the researcher himself/herself.
- **Benchmarking Problems.** This group concentrates on enhancing and improving known benchmark datasets results. These benchmarking datasets were developed in different academic institutions and made available to researchers and they have been recognized by most conferences and journals.

Benchmark datasets provide a balanced evaluation of the performance of the search algorithms used. The reason for that is that most benchmarking instances have a variety of developed tools that:

- Validate independently the accuracy of the obtained results.
- Provide a hardware dependency assessment through specification of the researcher's testing machine and calculates its maximum computational time to achieve a solution for that specific benchmarking problem.

In Appendix B, we present two of the most studies well-known benchmarking datasets for exam timetabling problem; the Toronto Benchmarking Datasets and International Timetabling Competition Benchmarking Datasets.

3.6 ITC 2007 Exam Timetabling Problem Description

ITC 2007 exam timetabling problem consists of the following main resources [127].

1. An exam session that can be attained in one or more periods over a pre-determined length of time. Number and length of Periods are provided.
2. A set of exams that are to be scheduled into periods.
3. A set of students registered in specific exams. Each student is registered in a number of exams. For each exam, the set of registered students is provided.
4. A set of rooms, each with its own provided capacity. Exams, however, can share rooms. Also an exam may take place in more than one room.

5. A number of hard Constraints which must be satisfied
6. A number of soft Constraints which, if they are violated, must subsidize to a penalty. Each soft constraint type has its own penalty cost.

The objective is to find a feasible timetable in which all examinations have been assigned to a period and room (Exam assignment) satisfying all hard constraints and minimizing violated soft constraints.

3.6.1 Hard Constraints

There are five types of hard constraints that must not be violated.

- There must not be any student who attends more than one examination at the same time.
- The capacity of each room must not be exceeded at any time throughout the examination session.
- Specific period constraints must be satisfied. For example, Exam 1 must take place after Exam 2.
- Specific room constraints must be satisfied. For example, Exam 100 must take place in Room A500.
- Period Lengths must not be violated.

3.6.2 Soft Constraints

Soft constraints are split into two categories: resource based constraints and global setting constraints.

3.6.2.1 Resource Based Soft Constraints

- Period related soft constraints: Some periods may have a constraint that sets the number of times that a period can be used. If it is exceeded, an associated penalty is multiplied by the number of exceeded usage. Different periods may have different associated penalties.
- Room related soft constraints: Likewise, some rooms may have a constraint that sets the number of times that a room can be used. If it is exceeded, an associated penalty is multiplied by the number of exceeded usage. Different rooms may have different associated penalties.
- Period spread of examinations: This is related to the number of times when students have to sit more than one exam in a time period specified by the institution. This is usually used as an indication of "fairness" principle to all students taking exams.
- Mixed duration of examinations within individual periods: The number of occurrences of exams timetabled in rooms along with other exams of differing time duration.

3.6.2.2 Global Setting Soft Constraints

- Two exams in a row: The number of events when students have to sit two exams in a row in the same day.

- Two exams in a day: The number of events when students have to sit two exams on the same day.
- Larger examinations appearing later in the timetable: This constraint is set by some institutions to penalize large exams that take place at a later stage in the whole exam session. This is because to avoid the issue of markers would take long time relatively to finish marking these exams and hence might delay announcing exam results. Both "large" and "later" terms are defined.

It is worth noting that it is up to the institution (that intends to solve its exam timetabling problem) to set soft constraints as settings aiming to get solutions that they see them as satisfactory to all involved resources, i.e. students, rooms, markers, lecturers…etc. This is why soft constraints are provided as weightings in the data instead of expecting them to be hard coded into the solution solver which gives a margin to set different weightings for each dataset.

3.7 ITC 2007 Exam Timetabling Benchmarking Datasets

This section presents the instances that we use as a case study in our testing for our proposed approach. We show how we model several real-life timetabling problems from ITC 2007 benchmark datasets. Next, we give an overview of the experiments we want to carry out. That is, we present the key questions we like to answer by means of the experiments. The benchmark dataset consists of 12 basic real

world examination time tabling problems obtained from anonymous different universities around the world. The detailed properties of the 12 benchmark instances are summarized in Table 3.

Instance #	# of students	# exams	# rooms	Period & Room Hard Constraints	Constraints density	# time slots
Instance 1	7,891	607	7	12	05.04%	54
Instance 2	12,743	870	49	14	01.17%	40
Instance 3	16,439	934	48	184	02.62%	36
Instance 4	5,045	273	1	40	14.94%	21
Instance 5	9,253	1,018	3	27	00.87%	42
Instance 6	7,909	242	8	23	06.13%	16
Instance 7	14,676	1,096	15	28	01.93%	80
Instance 8	7,718	598	8	21	04.54%	80
Instance 9	655	169	3	10	07.79%	25
Instance 10	1,577	214	48	58	04.95%	32
Instance 11	16,439	934	40	185	02.62%	26
Instance 12	1,653	78	50	16	18.21%	12

Table 3 - ITC 2007 12 Exam Benchmarking Datasets

The conflict density value is calculated using a formula taken from [52].

$$Conflict\ Density = \frac{\sum_{i=1}^{N-1}\sum_{j=i+1}^{N}(\frac{\beta_{ij}}{N})}{N}$$

Where N is the number of exams and β_{ij} is a decision variable assuming it is equal to value 1 if exam i and exam j have at least one student in common while assuming value 0 otherwise.

ITC 2007 only announced the first 8 datasets and made the last 4 hidden at the time of the competition. This is to measure the quality and performance of the contested solvers on problems that are not tested during the time of developing solvers.

Each instance of the exam track datasets contains detailed information about additional hard constraints that are concerned with periods and rooms specifically.

3.8 Soft Constraints Penalty Calculation

In ITC 2007 exam track, each data set contains a table of penalties for each violated type of soft constraints. There are five different soft constraints that when violated imply a penalty. The following is a summary on how these soft constraints violation penalties are calculated.

3.8.1 Two Exams in a Row

This penalty applies to exam assignments occurrences where two examinations are taken by a student, one straight after another, also known as back to back exams. It is calculated by totaling the number of students that violate the constraint and multiplied by the number provided in the “two in a row” weighting settings.

3.8.2 Two Exams in a Day

This penalty applies to exam assignments occurrences where two examinations are taken by students in the same day but are not directly back to back. This obviously conditioned by the fact that there must be three periods or

more in a day. The total number is consequently multiplied by the “two in a day” weighting settings.

3.8.3 Period Spread

This penalty concerns with how a student’s examination assignments are spread over the examinations periods. The higher the period spread number, the better the spread of examinations for individual students. The idea of this penalty is to penalize examinations that were assigned to the same student with the period spread setting. For example, if “Period Spread” is set at 6, for each exam we count all the occurrences of enrolled students who have to sit other exams afterwards but within “Period Spread” of 6 periods. This overall penalty is added to the overall penalty. It is notable that the same exams that contribute to this penalty could as well contribute to the previous two penalties “two exams in a row” and “two exams in a day” penalties.

3.8.4 Mixed Durations

This penalty depends on penalizing assignments based on rooms and period and not exams where there are mixed durations. The purpose of the penalty is to make sure that exams which happen to take place at the same time should be of the same length or otherwise carry a penalty that will be added to the total penalty.

3.8.5 Front Load Penalty

Most institutions desire that examinations with the largest numbers of students are scheduled at the beginning of the examination session so that markers would be under no stress and would take their time in marking exams. The penalty for this concept in ITC 2007 is called "Front Load" and is defined as a sequence of three parameters n, m, t. The idea behind this penalty is to allow the institution to try to schedule larger exams earlier in the examination session.

The first parameter defines how the largest exam is defined in terms of the number of students. If the number is, for example, 200 then any exam that has enrolled students of 200 or more is considered to be one of the largest exams. The second parameter is the number of last periods that these larger exams should be avoided to be scheduled in. The third parameter is the penalty or weighting that should be added each time the constraint is violated. For example, if "Front Load" = 200, 20, 15, It means that any exam with 200 students or more that is scheduled in the last 20 periods will be penalized with 15.

3.8.6 Room Penalty

This penalty is about keeping using certain rooms. For each period, if a room used within the solution has an associated penalty, the penalty for that room for that period is calculated by multiplying the associated penalty by the number of times the room is used.

3.8.7 Period Penalty

As the case with rooms, it is often the case that institutions desire to keep the usage of certain periods to a minimum. For each period, the penalty is calculated by multiplying the associated penalty by the number of times the exams timetabled within that period.

Table 4 shows the different penalties for violating soft constraints for the 12 datasets.

Instance #	Two in a Row	Two in a Day	Period Spread	Mixed Durations	Front Load
Instance 1	7	5	5	10	100, 30, 05
Instance 2	15	5	1	25	250, 30, 05
Instance 3	15	10	4	20	200, 20, 10
Instance 4	9	5	2	10	050, 10, 05
Instance 5	40	15	5	0	250, 30, 10
Instance 6	20	5	20	25	025, 30, 15
Instance 7	25	5	10	15	025, 30, 10
Instance 8	150	0	15	25	250, 30, 05
Instance 9	25	10	5	25	100, 10, 05
Instance 10	50	0	20	25	100, 10, 05
Instance 11	10	50	4	35	400, 20, 10
Instance 12	35	10	5	5	025, 05, 10

Table 4 - ITC 2007 Exam Timetabling Data Sets' Soft Constraints Violation Penalties

3.9 Summary

This chapter has detailed the examination timetabling problem in terms of definition, models, formulation and current research state. Based on what was presented in this chapter we were motivated about designing our solver and approach to solve the problem of large exam timetabling problem. It also introduced

ITC 2007 Benchmarking datasets. In Appendix B, a study on the different benchmarking datasets that are used by most researchers is detailed.

The work presented in this and previous chapters provide the necessary background and foundation for the hybrid metaheuristic approach to solve large examination timetabling problems that we describe in more details in the next chapter.

4. Solving the Teaching Assignment Problem

In this chapter, the problem of teaching assignment is addressed. It involves the assignment of teachers to already scheduled courses to time slots. We formulate the problem as a COP (CSP with simple soft constraints). A modified backtracking with look ahead algorithm to handle soft constraints is proposed for solving the problem. The performance of the proposed algorithm is evaluated using two sets of real data and some randomly generated problem instances.

4.1 Introduction

In this chapter, we describe a constraint programming system, with a website as front-end to demonstrate a suggested solution technique of the Teaching Assignment Problem. As stated in the previous chapter, the timetabling problem in general is mostly, if not always, an over constrained combinatorial optimization problem and hence it is considered one of the most difficult problems to solve.

Teaching Assignment Problem in essence is a branch of the timetabling problem and it is the problem of assigning professors to timeslots that is occupied by courses in a specific week. The resulted weekly timetable is to be used to organize the teaching process at a university or any educational institute given that the courses have already been scheduled over the timeslots and rooms. Each professor is assigned a maximum total number of courses to be taught that should not be violated. Each

professor is allowed to express interest or dislike in certain courses through weighed preferences that can or cannot be satisfied. Some professors can be assigned some courses in advance.

The final solution should be constructed based on distributing the given courses over professors based on fairness principle as well as maximising the total weight of the solution or minimizing the total cost of the solution, whichever was adopted as an approach for soft constraints. The total weight/cost of a solution is the sum of all satisfied preferences.

The literature is very rich on the topic of university timetabling in general as there are different ways to solve the problem; most of them depend on specific needs considered by the institution that the timetabling is designed for. However, to the knowledge of the authors, no literature is dealing with the teaching assignment problem.

In our case, we considered the problem as two-fold stages. The first is to assign courses to rooms and timeslots and the second is to assign professors to the resulting timeslots with courses. In this study, we only tackled the second one.

Timetabling problems, in general, are usually over constrained as it is not always possible to satisfy all requirements. User preferences can be used to relax these requirements. In our study case, we have used a more specific model with preferences

which utilize weight for each constraint and try to maximize total weight of satisfied soft constraints.

As a development approach, our work includes a development of a solver for soft constraints. The solver was implemented as an extension of a well-known CSP solver named “Java Cream” [177] to include soft constraints in the backtracking mechanism which the Java Cream Solver was lacking. The solver itself was re-coded entirely, by the author, using Microsoft C# language from Java language. Some of the optimized technologies introduced in C# and in .NET framework, such as LINQ, were used to enhance and optimize the local search.

Language-Integrated Query (LINQ) is a set of features in Microsoft© .NET framework that extends powerful query capabilities to the language syntax of C#. LINQ introduces standard patterns for querying data, and the technology can be extended to support potentially any kind of data store. Besides, LINQ queries provide two main advantages over traditional loops: They are more brief and comprehensible, especially when filtering multiple conditions, and they provide powerful filtering, ordering, and grouping capabilities with a minimum of application code. In general, the more complex the operations wanted to carry out on the data, the more benefit will be achieved by using LINQ instead of traditional iteration techniques.

We have used data from the department of Computer Science at the University of Regina as a case study. At the time of writing this chapter, the problem is handled by the department using manual methods.

The next section of this chapter provides a related work for the problem. Section 3 provides a description to the teaching assignment problem. The added soft-constraint approach that was implemented within the solver along with the modified search algorithm developed for this problem is detailed in section 4. This includes a description of how the problem has been solved as well as the representation of soft and hard constraints. Furthermore, a discussion to the method on how the search is done is provided at the end of this section. Section 5 provides a description for the web based system used to implement the solver. Computational results are discussed in Section 6. The final section reviews the results of two experiments and testing that we carried out of our approach and looks to future extensions of the problem solution and soft-constraint solver improvements.

4.2 Related Work

The timetabling problem is considered to be one of the broadly studied scheduling problems in Artificial Intelligence and Operations Research literature [185]. Educational timetabling, to be specific, has been the main topic of quite few papers in various scientific journals and the topic of many theses in academia society. The course timetabling problem deals effectively with courses and timeslots which have to be scheduled during the academic term. The problem basically is the

scheduling of a known number of courses into a known number of timeslots spread all over the week in such a way that constraints are satisfied.

As there are many versions of the Timetabling Problem, a variety of techniques have been used to solve it [22], [55]. Most of these techniques vary from graph colouring to heuristic algorithms.

Another focus of research in the timetabling problem was on the application of a single solution approach which in effect a large variety of such approaches have been tried out, such as an integer programming approach [55], Tabu search [22], and Simulated Annealing [191]. Recently, some researchers have attempted to combine several approaches, such as hybridization of exact algorithms and Metaheuristics.

One of the most primitive methods used to solve this problem is graph colouring in which vertices represent events where two vertices are connected if and only if there is a conflict. [191], [199], [173], [27] and [132] proposed a number of formulations by graph colouring for a set of class teacher timetabling problems and discussed the inherent complexity.

In [183], graph colouring has been used to solve course and exam timetabling. Linear programming models were also used to formulate the course time-tabling problem usually with binary variables [68], [67], [80], and [79]. An Integer Programming approach [172] was also used to model the timetabling problem as assignment problem with numerous types of constraints and large number of binary or integer

variables. Rudov and Murray introduced an extension of constraint logic programming [164] that allows for weighted partial satisfaction of soft constraints is implemented to the development of an automated timetabling system.

In [89], an Evolution Strategy to generate the optimal or near optimal schedule of classes is used to determine the best, or near best timetable of lecture/courses for a university department.

Case Based Reasoning is another approach that has recently been applied to university timetabling [35], [34], [19], and [44]. Case Based Reasoning is believed to be studied as early as 1977 with the study of Schank and Abelson [170]. Case Based Reasoning has also been successfully applied to scheduling and optimization problems. Burke et al. [45] also, in a published article, developed a graph-based hyper-heuristic (GHH) which has its own search space that operates in high level with the solution space of the problem generated by the so-called low level heuristics.

4.3 Problem Description

In general, the timetabling problem is the assignment of timeslots to a set of events. These assignments usually include many considerable constraints of different types. At the department of Computer Science, University of Regina, in any term, the timetabling process currently consists of constructing a class schedule prior to student registration. The professors and classes timetabling problem [92] and [91] is NP-Complete. The teaching assignment problem is the problem of assigning courses,

scattered over timeslots, to professors. In our case, the teaching assignment problem described as follows:

- There is a finite set of courses $C = \{c_1, c_2, \dots, c_{|C|}\}$ and a finite set of timeslots $T = \{t_1, t_2, \dots, t_{|T|}\}$, which already have been assigned to courses C. This is typically provided as courses occupy timeslots. So each course could occupy just one timeslot, usually 3 hours; two timeslots, usually an hour and half each; or 3 timeslots, usually an hour each. For any course, the timeslots assigned to it must not overlap. t_i can be assigned to different courses as long as they are in different rooms and different professors. Our approach is nothing to do with these assignments as these assignments are considered as input for the problem to solve.
- There is a finite set of professors $P = \{p_1, p_2, \dots, p_{|P|}\}$.
- In this scheduling problem, courses represent variables while professors represent variables domain values.
- The problem is to schedule P to C in a way such that no professor p_i is in more than one place at a time t_i.
- The constraints for this problem are soft and/or hard. The soft constraints might not be satisfied and on the contrary hard constraints must be satisfied so a possible solution to the problem is one that satisfies all the hard constraints but not necessarily soft constraints.

- Soft Constraints are preferences that do not deal with time conflicts and have weight (or Cost) associated with them. Our goal is to maximize the total weight of a solution (or minimize the total cost). We have two types of soft constraints; the first is count, where professor has a maximum number of courses assigned to him that should not be exceeded. The second is preferences that any professor can express as interest or dislike in certain courses which have weights (or costs). This type of constraints can or cannot be satisfied.
- Hard Constraints are typically constraints that physically cannot be violated. This includes timeslots that must not overlap in time, and in our problem timeslots that overlap in time must not be taught by the same professor. There is another type of hard constraints where timeslots represent a course can be assigned to a professor in advance prior to starting the search for a solution.
- There is a total weight function that measures the quality of the current solution. The object of this function is to return the sum of all weights/costs associated with the satisfied preferences. The aim of the optimization technique is to maximize the total weight function or minimize the total cost.

4.4 Proposed Solution

As mentioned above, the solver, used in solving the problem, is re-coded from a well-known solver named *"Java Cream"* using Microsoft C# language. The original solver can be used to model any constraint satisfaction or optimization on finite

domains problems [176]. However, it lacks any proper handling of soft constraints. In essence, the original solver handles only hard constraints and hence the problem has to be modelled as CSP. As known, any timetabling/scheduling problem would be mostly over constrained and therefore it cannot be solved unless some constraints are relaxed. Hence, the necessity came to add soft constraints as part of the re-coded solver to solve timetabling problems. The backtracking method is the one that was modified to take into account soft constraints. In our approach, we needed to model the problem as COP so that we can handle soft constraints appropriately.

Although we describe only the modified backtracking method, which is used by two of the five methods that the solver adopts; Branch and Bound, Iterative Branch and Bound. However, because preliminary tests showed that performance is not significantly improved in our application when using the other three methods; Taboo Search, Random Walk and Simulated Annealing, we only consider the backtracking method to meet our requirements. Furthermore, all solver search methods use the same variable/value ordering methods and hence would use the same approach mentioned here. We think that because the first application study case variables and values are relatively small, the tests performance using other methods has not improved but might be better if another application is implemented which might have more complex variables and values. However, in the second application case study where we randomized the number of variables and values to be large, we consider

the testing on the four of the solver's methods to have tested the full extent of our approach.

The backtracking method is the one that was modified to take into account soft constraints. Basically, the backtracking algorithm [72] has two phases. The first stage is what is called "a forward phase" in which the variables are selected sequentially and the current partial solution is extended by assigning a consistent value for the next variable if one exists. The second phase is known as "a backward phase" in which the algorithm returns to the previous assigned variable when no consistent solution exists for the current variable.

For the forward phase, the adopted solver originally decides which variable to instantiate next by selecting the one that has minimum number of domain values (i.e. the one that its domain size is minimum). Then the solver decides which value to assign to the next variable by assigning the maximum value in the variable domain. By assigning a value to a variable, this value is eliminated from all other variables' domains. This is known as look-ahead backtracking.

Because of the soft constraints that have been added to the solver, we have improved the two backtracking phases as followed if "soft constraint approach" method is selected in solving the problem:

- On deciding which variable (Course) to instantiate next, the solver tries primarily to select the variable with the highest weight on equal soft

constraints that has not been assigned a value (Its domain size is greater than one); if not then it will return randomly one of the variables (courses).

- The idea behind this is to try to select a variable that has soft constraints associated with it first, if not found then it will act on the other types of variables.
- If the previous hint is not implemented, it will give the chance to assign values to variables that do not have soft constraints with them where they should have been at least trying to be assigned to variables with soft constraints. In this case variables with preferences will miss the chance to get their preferences assigned to them.
- On deciding which value to assign to the variable selected in the previous step, a method, first, checks if there are equal soft constraints associated to that variable. If there are not any, then it will randomly select a value from its domain (i.e. domain values represent professors).
- It is worth mentioning that even if there are no soft constraints associated with it, the method tries to not to choose a value that is associated with another variable that has an equal soft constraints as it might be needed in a later stage.
- If there are indeed equal soft constraints associated to that variable, then it assigns the value that least has been assigned to any variable before. This is in compliance with the "fairness" principle. Furthermore, the value is selected randomly if there is more than one value.

4.5 Web Based Interface

We have implemented a web-based application for solving the timetabling problem. The idea behind this approach is to get professors to enter their preferences through the web site. Web based applications generally are more convenient for users. For instance, every professor can enter his/her preferences from office/home and there is no need to provide this information to application operator to enter their data.

The Teaching Assignment Problem Solver (TAPS) was developed using Microsoft ASP.NET MVC (Model-View-Controller) as an interface, Microsoft SQL Server 2005 as database engine and Internet Information Server (IIS) as web server. The MVC model provides a rich graphic user interface using HTML and JQuery (Java script based library). There are four main parts for the web site. The first is devoted to courses management and its information can be entered by administrator. The second is dedicated to professors' information management and their information can be entered by professors themselves. The third is for searching for a solution using the information provided and using the C# Cream as a background solver. The last section is for website settings. This includes application settings for courses and professors.

4.5.1 Course Section

The course section provides an interface for entering courses information. The information includes the course assigned week days, assigned timeslots, and assigned professor (if needed). The later will be treated by the solver as hard constraint. The

interface also displays the number of professors interested and not interested in that course. The whole courses table is displayed as a table where there is an option for inserting, editing and deleting a course.

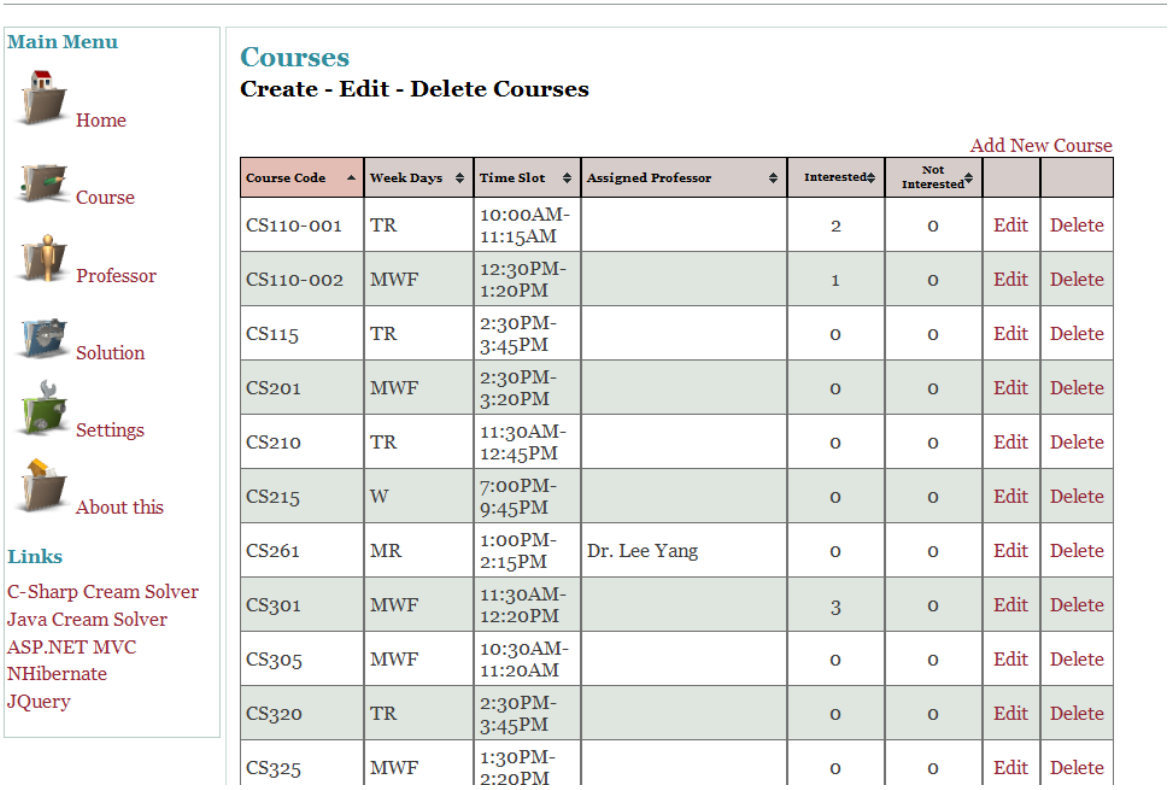


Figure 2 - Courses Screen

4.5.2 Professor Section

The professor section provides an interface for entering professors information. The information includes the number of courses that can be assigned to each professor (i.e. count constraint) and the professors' preferences. Both preferences will be dealt by the solver as soft constraints.

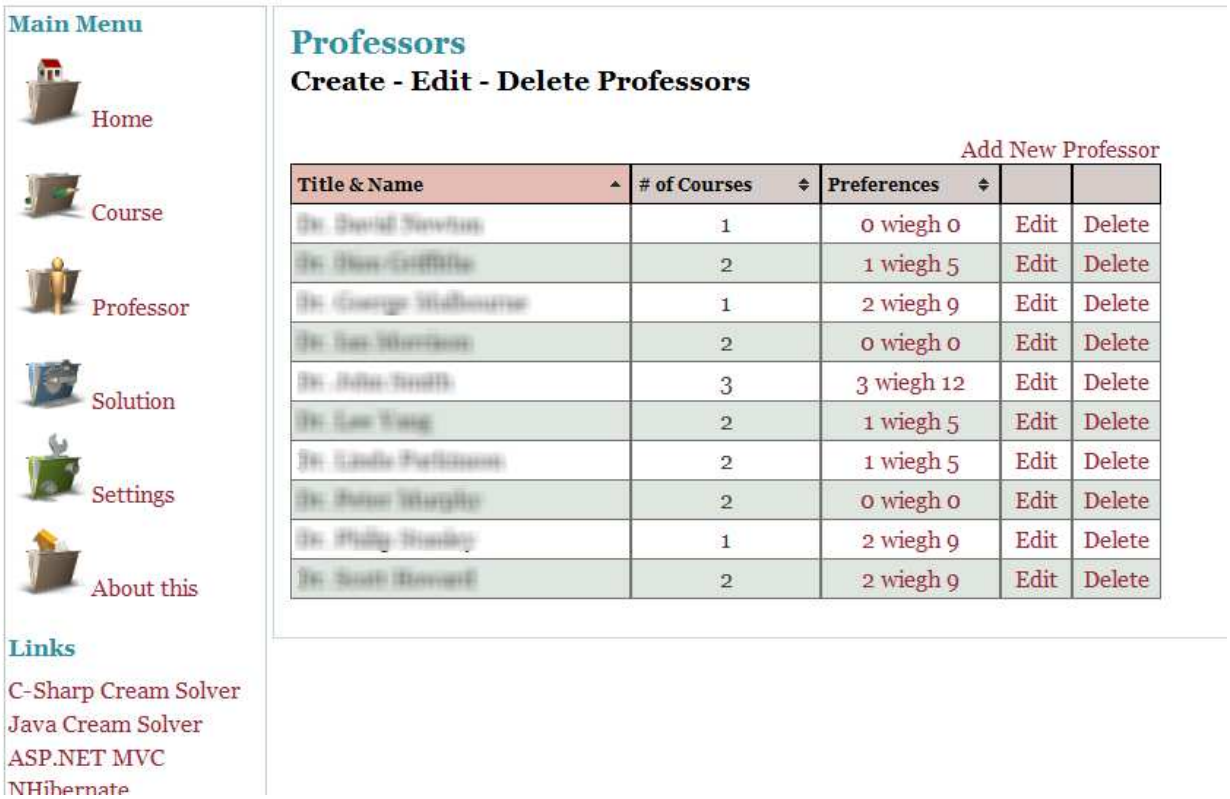


Figure 3 - Professors Screen

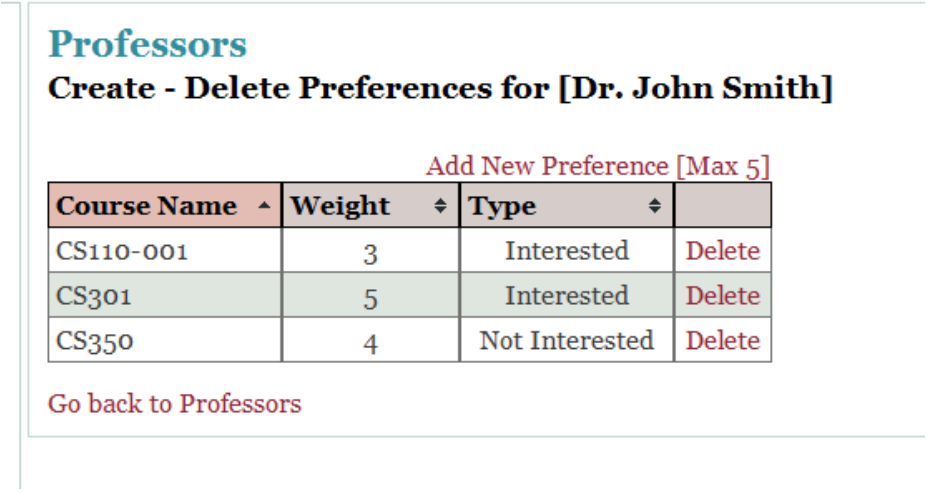


Figure 4 - Professor Edit Screen

4.5.3 Solution Section

The solution section provides an interface for searching for solutions using the information provided in the previous two sections. If solutions found, they will be displayed in this section's web page. Among the information displayed, there is time spent for generating all solutions and the time spent for each solution along with the solution weight. There is also the option to display the next and previous solution. There are three tables; the first one is the main table where courses are displayed with professors. The second table displays professor's names and the number of assigned courses. The displays all constraints (hard and soft) used in finding the solutions.

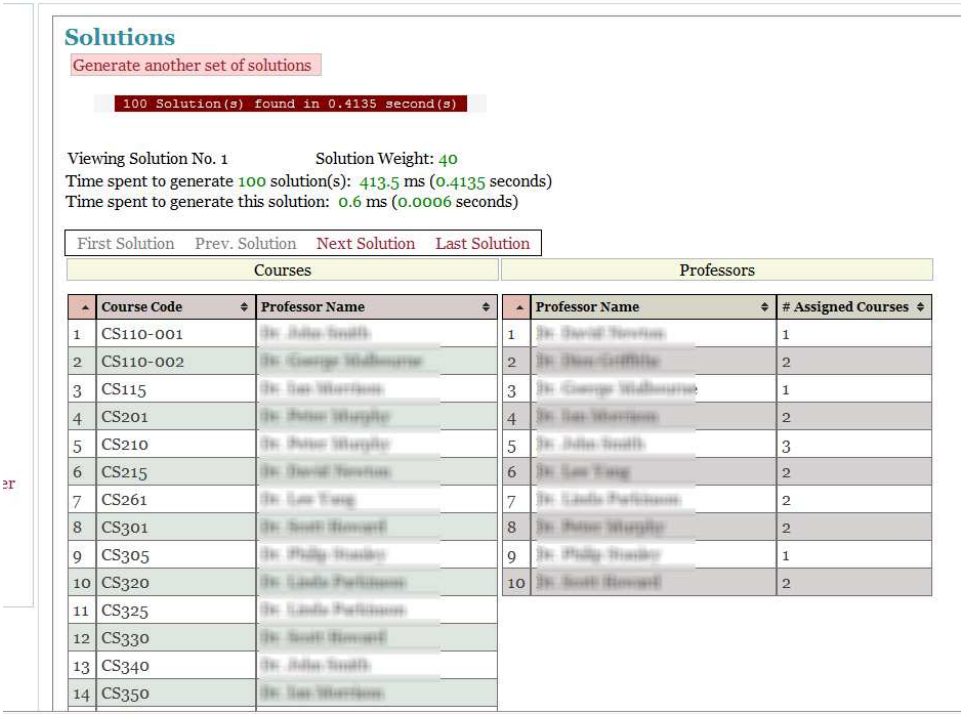


Figure 5 - Solutions Screen

4.5.4 Settings Section

The settings section provides an interface for entering solver settings. This includes the number of hours per course, maximum break minutes per session, maximum number of courses per professor, number of preferences per professors, maximum number of generated solutions, the option to generate only better solutions in terms of solution weight and the maximum timeout that should be used in the solver to generate solutions. The rest are self-descriptive from the following figure.

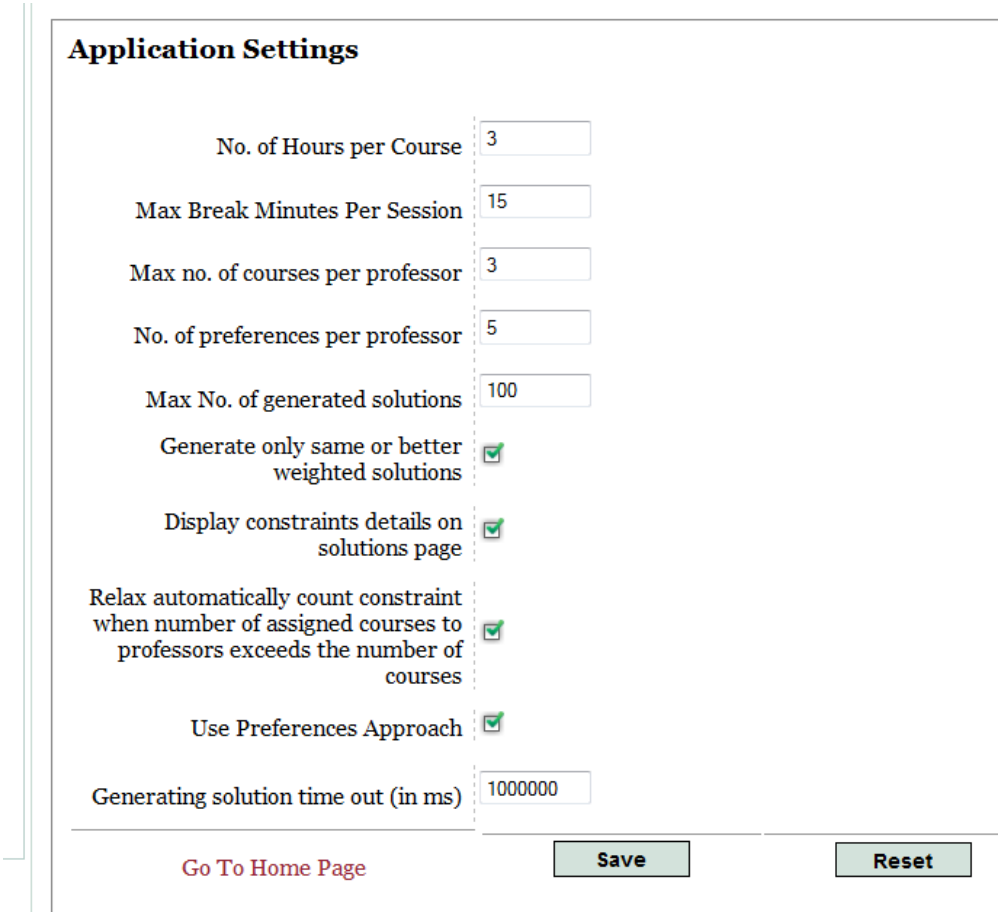


Figure 6 - Settings Screen

4.6 Experimental Results and Evaluation

Two types of experiments were conducted in order to justify our proposed solution. The first experiment is concerned with the enhanced technique that uses soft constraints against the original technique that uses only hard constraints. The second experiment is to test and compare four or the five methods that the solver includes.

4.6.1 Experiment I

The first experimental test is to compare between our proposed approach and the original backtracking method [160].

In order to do tests on the designed website and the proposed solver, we used data from the Computer Science department at the University of Regina, for both courses and professors information including courses timeslots. Then we assigned randomly some of the courses to some professors and assigned professors to show some interest and dislikes in some of the courses.

Overall, we used 17 courses as solver variables and 10 professors as solver values. From these courses we entered interest in 4 courses for different professors and disinterest in just one course. Experimental computations were done with a number of objectives in mind. The main goal was to provide a table of courses assigned to professors where all hard constraints are satisfied and the weight of soft constraints is maximized.

The second experiment involved using the same solver to solve a problem with the same variables and domain values but using non preference approach (The original backtracking method). We have also set the solver to generate the first 100 solutions considering the first solution is the most optimized one and to generate only same or better weighted solutions and the solver has only 100 seconds to generate any solution at any given time. We used a PC with the following capabilities: Core 2 Duo Quad processor (2.4 GHz) with 6 GB ram. We have asked the solver to search for solutions 10 times to get a bigger picture of the search time spent in finding solutions and here are the results:

Attempt#	**average time (ms)**	Feasible solution using preferences	Feasible solutions not using preferences
1	**2.223**	Within the first 10 solutions	Not generated within the first 100 solutions
2	**1.920**	Within the first 10 solutions	Not generated within the first 100 solutions
3	**2.647**	Within the first 10 solutions	Not generated within the first 100 solutions
4	**2.220**	Within the first 10 solutions	Not generated within the first 100 solutions
5	**2.193**	Within the first 10 solutions	Not generated within the first 100 solutions
6	**2.454**	Within the first 10 solutions	Not generated within the first 100 solutions
7	**2.567**	Within the first 10 solutions	1 within the last 10 solutions out of 100
8	**2.062**	Within the first 10 solutions	Not generated within the first 100 solutions
9	**2.424**	Within the first 10 solutions	Not generated within the first 100 solutions
10	**2.168**	Within the first 10 solutions	Not generated within the first 100 solutions
Average	**2.2879**	Within the first 10 solutions	1 within the last 10 solutions out of 100

Table 5 - TAP Experiments I results

When using non-preference approach, we were unable to find a feasible approach that can satisfy the maximum number of soft constraint in the first 100 solutions. On

the contrary, when using the preference approach, the first 10 solution were optimal for our problem that satisfied hard and soft constraints.

4.6.2 Experiment II

In order to compare the solver's four methods that the CSharp Cream Solver adopts (Branch and Bound, Iterative Branch and Bound, Taboo, Random Walk) to solve the Teaching Assignment Problem, an experiment has been conducted that involved developing a piece of code to generate a number of courses, a number of professors and a number of soft constraints and hard constraints. The generation of the courses was to be of 3-hour periods distributed either by one session of 3 hours, 2 sessions of an hour and a half or 3 sessions of an hour each. Of course, the generation of random courses will result of not equal constraints due to time conflicts.

The experiment generated 10, 20, 30, 50, 80, 100, 200, 250, 300, 500, 800 courses consequently. So the first time the application will generate 10 courses and the corresponding professors and hard and soft constraints; the solver then will try to find just the first solution using the four methods. The second time will use 20 courses and the third time will use 30 courses and so on. In each of these sessions, the number of professors is always 33% of the number of courses. The solver has 10 minutes (which is relatively a long time) before it times out. Also, a random number of soft constraints were generated that expressed by professors in the courses as well as

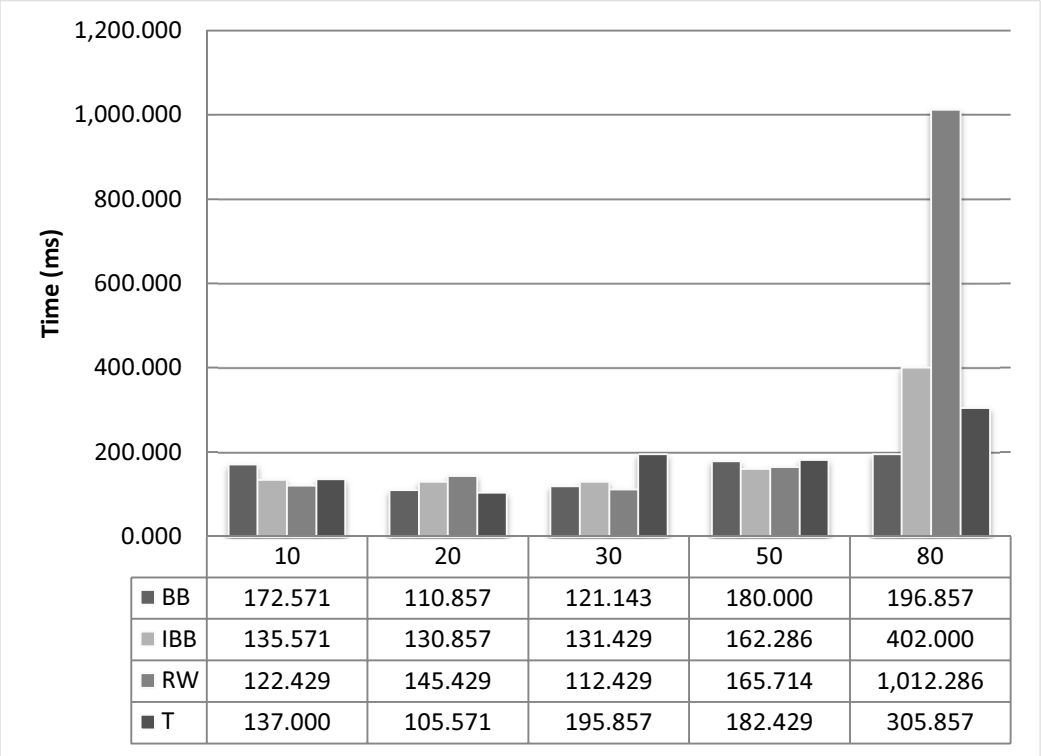
random count constraints assigned to each professor. It is worth mentioning that the courses represent solver variables and professors represent variables values.

The solver was set up to run 7 times for each session (for each number of courses "variables") to get the average for better results.

Here are the findings:

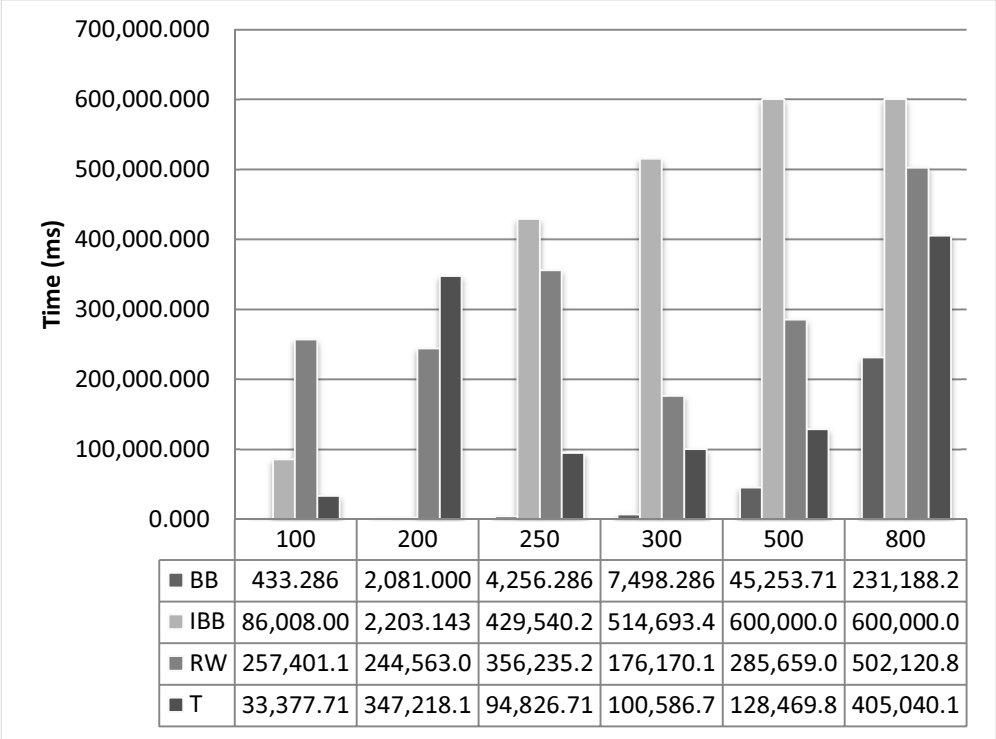
- Branch and Bound always can find a solution and in most cases is better than other methods especially if the number of variables is high.
- For the number of variables from 10 to 80 all methods generated a solution almost in similar times.
- For the number of variables from 10 to 80, none of the used methods failed to find the solution in all seven attempts.
- Starting from the number of variables of 100, methods apart from Branch and Bound started to fail to find a solution in at least 1 out of the 7 attempts failed to find a solution.
- The average success percentage for branch and bound was the best among the four methods which staggeringly reached 100%. Taboo came second with 82.14%, Random Walk came third with 67.86% and Iterative Branch and Bound came last with 53.57%.
- Iterative Branch and Bound failed to find any solution in all 7 attempts for the number of courses between 500 and 800.

- For the number of variables of 800, Taboo method failed in 3 out of 7 attempts to find a solution while Random Walk failed in 5 out of 7 attempts.



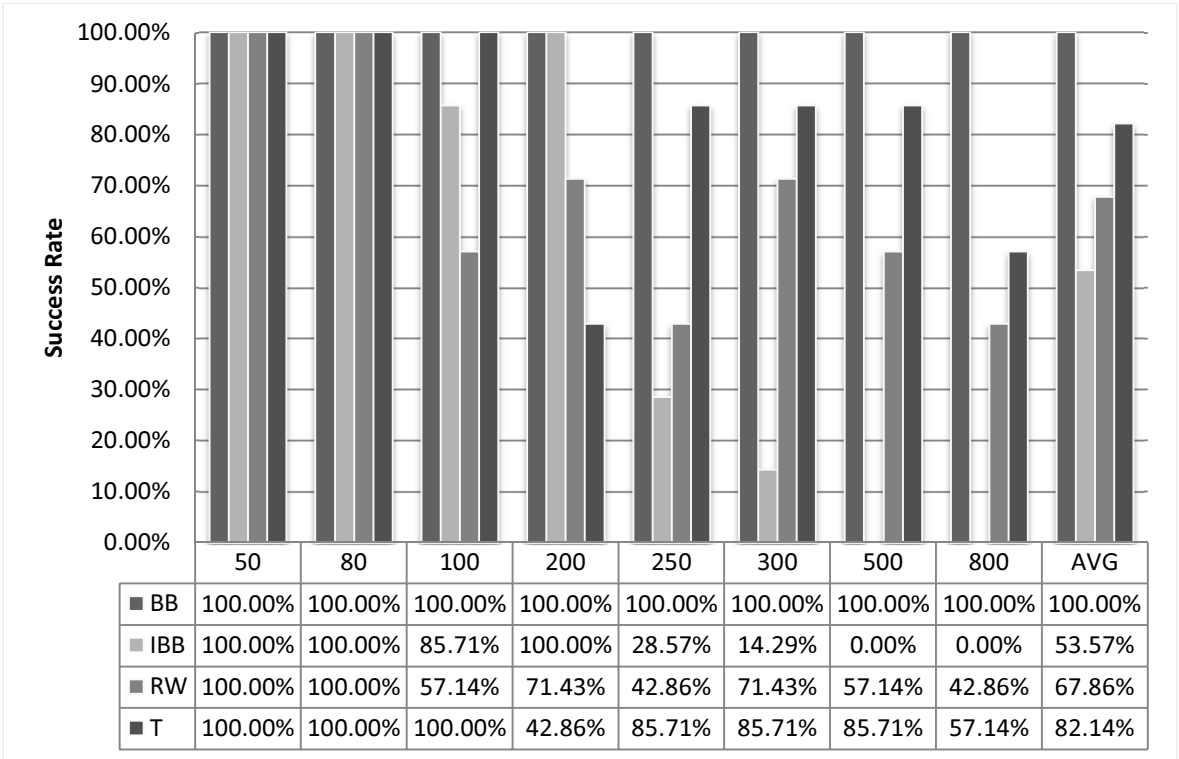
	10	20	30	50	80
BB	172.571	110.857	121.143	180.000	196.857
IBB	135.571	130.857	131.429	162.286	402.000
RW	122.429	145.429	112.429	165.714	1,012.286
T	137.000	105.571	195.857	182.429	305.857

Figure 7 - Solver methods average times for the number of courses of 10 to 80



	100	200	250	300	500	800
BB	433.286	2,081.000	4,256.286	7,498.286	45,253.71	231,188.2
IBB	86,008.00	2,203.143	429,540.2	514,693.4	600,000.0	600,000.0
RW	257,401.1	244,563.0	356,235.2	176,170.1	285,659.0	502,120.8
T	33,377.71	347,218.1	94,826.71	100,586.7	128,469.8	405,040.1

Figure 8 - Solver methods average times for the number of courses of 100 to 800



	50	80	100	200	250	300	500	800	AVG
BB	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
IBB	100.00%	100.00%	85.71%	100.00%	28.57%	14.29%	0.00%	0.00%	53.57%
RW	100.00%	100.00%	57.14%	71.43%	42.86%	71.43%	57.14%	42.86%	67.86%
T	100.00%	100.00%	100.00%	42.86%	85.71%	85.71%	85.71%	57.14%	82.14%

Figure 9 - the solver's four methods average success percent

4.7 Conclusion

We have successfully applied modified back tracking method to solve the teaching assignment problem. The problem was modelled as a COP problem. Feasible schedules were obtained for real data sets, including professors' preferences without the need for a huge computational effort. The original solver was meant to solve integer based variables problems but for problems with hard constraints. We have extended the solver to adopt soft constraints and we think that it has been a success. In conclusion, this application of Teaching Assignment Problem Solver appears to be quite successful and we are satisfied and ready to implement it to generate actual schedules for future terms. We also think that this approach can be implemented in similar problems like Exam Supervision scheduling. Based on the gathered experience of this test, we concluded that this approach is computationally feasible.

5. Incremental Dynamic Search Solver

This chapter introduces Incremental Dynamic Search (IDS) Solver, a local-search based solver, we propose, for solving general finite constraint satisfaction and combinatorial optimization problems. The goal of the IDS Solver is to offer a pluggable approach to COP models and to be used in the remainder of the thesis as an experimental platform for composing a framework for parallel local search extension, whose functionality is determined by the modelled problem. The solver tries to improve the efficiency, the robustness, and the usability of local-based search algorithms.

In this chapter, the whole solver along with the parallel extension (written in Microsoft© C#) is presented.

5.1 Introduction

In combinatorial optimization, the Systematic Search Algorithms (Tree Search Algorithms) explore the search space by iteratively instantiating variables putting together an optimal solution. The efficiency of these techniques is based on the chance to prune the tree search, which has, in worst case, an exponential size. Integer Programming (IP), which was originated from tree search techniques, is one of the most dominant tools of operations research. The technique was so successful among operations research practitioners due to its application simplicity, even though it is imperfect in solving large-scale combinatorial problems.

The reason for this imperfection is due to the belief that the mathematical formulation of large problems increases the number of variables and constraints to an extent that becomes huge for practical problems [51]. However, there were some efforts to reduce the size of the problem proposed in the literature. For example, Lennon in [120] in regard to modelling and solving large examination timetabling problem, reduced the search space by utilizing Integer Programming only to schedule the most difficult exams (in terms of associated constraints). Following that, a heuristic is used to schedule the remaining exams.

Based on the success of IP programming solvers, Constraint Programming (CP) solvers were purposed as a "model-and-run" approach. If effectively applied, it can ease significantly the development and maintenance stages of solving optimization problems.

Local Search techniques, on the other hand, are based on applying iteratively some changes, named moves, to a complete assignment (or a solution is not necessarily feasible) to improve an objective function. Although incomplete as it does not guarantee to find an optimal solution, the technique is widely adopted because it allows achieving good quality solutions in relatively minimal iterations. However, the problem of designing and implementing local-search algorithms is not straightforward. This is because the algorithm's function that is responsible for

making moves decisions is particularly difficult to design, as it requires specific expertise in algorithms and skills in computer programming [3].

This chapter introduces IDS Solver, a local-search based solver for general CSPs and COPs. IDS solver allows modeling of a problem using constraint programming primitives; variables, values and constraints. It also allows implementers to focus on the design and modeling of the problem using a simple and pluggable approach that rely on the base solver to determine its final resolution based on efficient and reliable local-search techniques.

Although IDS solver is based on concepts of local search methods, there is an essential dissimilarity. Unlike classical local search which only works by iteratively improving a complete assignment over the variables which is not necessarily a feasible solution, IDS can operate over complete or incomplete feasible assignment. In such assignment, similar to backtracking based algorithm, all hard constraints that involve assigned variables must be satisfied but still some variables can be left unassigned. The idea is to increment the feasibility value of the incomplete solution during the search instead of working on infeasible complete solution. This proves to be advantageous when it comes to optimize and solve large combinatorial problems. The dynamic feature of the solver is to describe the fact that it can be easily start, stop, restart or continue from a feasible assignment, whichever complete or incomplete. This feature establishes the main functionality of the parallel solver as we will discuss.

In chapter 2 we highlighted systematic search and local search algorithms, heuristics and techniques. We also went through the different techniques for solving CSPs and COPs and usually one of three main families is used: local search algorithms and systematic algorithms, or hybrid approach. Each family has its advantages and disadvantages.

5.2 Incremental Dynamic Search Solver

The idea of IDS was born based on some of the above mentioned ideas. In the real world, there are many well-known widely-accepted software tools that are available for solving constraints satisfaction problems like Choco [58] and ILog Optimizer [106]. Some of these solvers are white boxes and most are black boxes. In software engineering "White Box" is usually a subsystem which its internal implementation is open and can be viewed but can only be modified upon feature request. This is turned out to be very helpful during white box testing where the system is fully examined to verify that it fulfills its requirements. In some cases, a white box system is open source software where its source code made available to public for extension (in an informal way), Choco is an example. On the contrary, a black box is a system which can only be viewed in regard to its input, output and system function without any proper knowledge of its inner implementation. A "Black Box Solver" is a solver that its source code usually is not available and usually is a commercial solver, ILog optimizer is an example.

Local Search research-level prototypes and frameworks are few or otherwise have gained limited degree of acceptance. The reasons for this, in our view, are twofold: first, the obvious straightforwardness and easiness of Local Search techniques which, ironically, encourages users and researchers to build their own applications from scratch but using problem-specific modelling. In contrast, the second factor is that the rapid evolution of Local Search techniques appears to create the impression that it is unfeasible and unrealistic to develop generic solvers that fit any problem.

It is our belief that the use of object-oriented frameworks and software's best practices can help in overriding these problems. A framework, essentially, is a guidance library that involves some sort of a hierarchy of abstract classes and interfaces. The user, developer or implementer should only define appropriate derived classes, which implement some virtual methods of the abstract classes and apply some contracts that conform to interfaces methods. The framework, therefore, should be responsible for the overall control structures for the essential and invariant parts of the algorithm, and it is the responsibility of the user or implementer to provide the details of any specific problem and how it should be modelled by overriding and overloading classes and methods as required.

In this chapter we present our attempt to develop a general solver for developing and the analyzing CSPs and COPs using Local Search algorithms. The system is called Incremental Dynamic Search Solver (IDS Solver), and it was fully built using object-

oriented frameworks and software best practices using C# as a development language and Microsoft .NET as a core framework. The solver we are proposing is easily extendable and pluggable and completely configurable. It has also a unique feature that makes it stand out of all solvers which is it can operate optionally in a parallel environment locally or across a network using Message Passing Interface concepts and frameworks. This feature's implementation is fully covered in Chapter 6 in solving large examination timetabling problems. The concept behind the name comes from the fact that it increments the feasibility value of the solution sought and it dynamically can start from an initial assignment, stop at any point and restart from any pre-saved solution.

5.2.1 Solver Layers

In software engineering, a layer is a logical arranging mechanism that organises the components that structure a full software component. Multi-layer architecture is a well-known approach in software design. This practice is followed so that it makes each layer in-dependable on others as well easy to maintain. The solver's framework consists of four main layers that communicate with each other to achieve different tasks. In our solver, the flow control is to be passed from bottom-up. Figure 10 shows the full hierarchy layers of the entire solver. Two main parts divides the solver categorically into problem-dependent and problem-independent parts. The problem independent part was developed to adapt to solving any CSP or COP as long as the problem is modelled to inherit and conform to the solver's main

classes while the problem-dependent part is specific to each problem in terms of input, configuration and modelling.

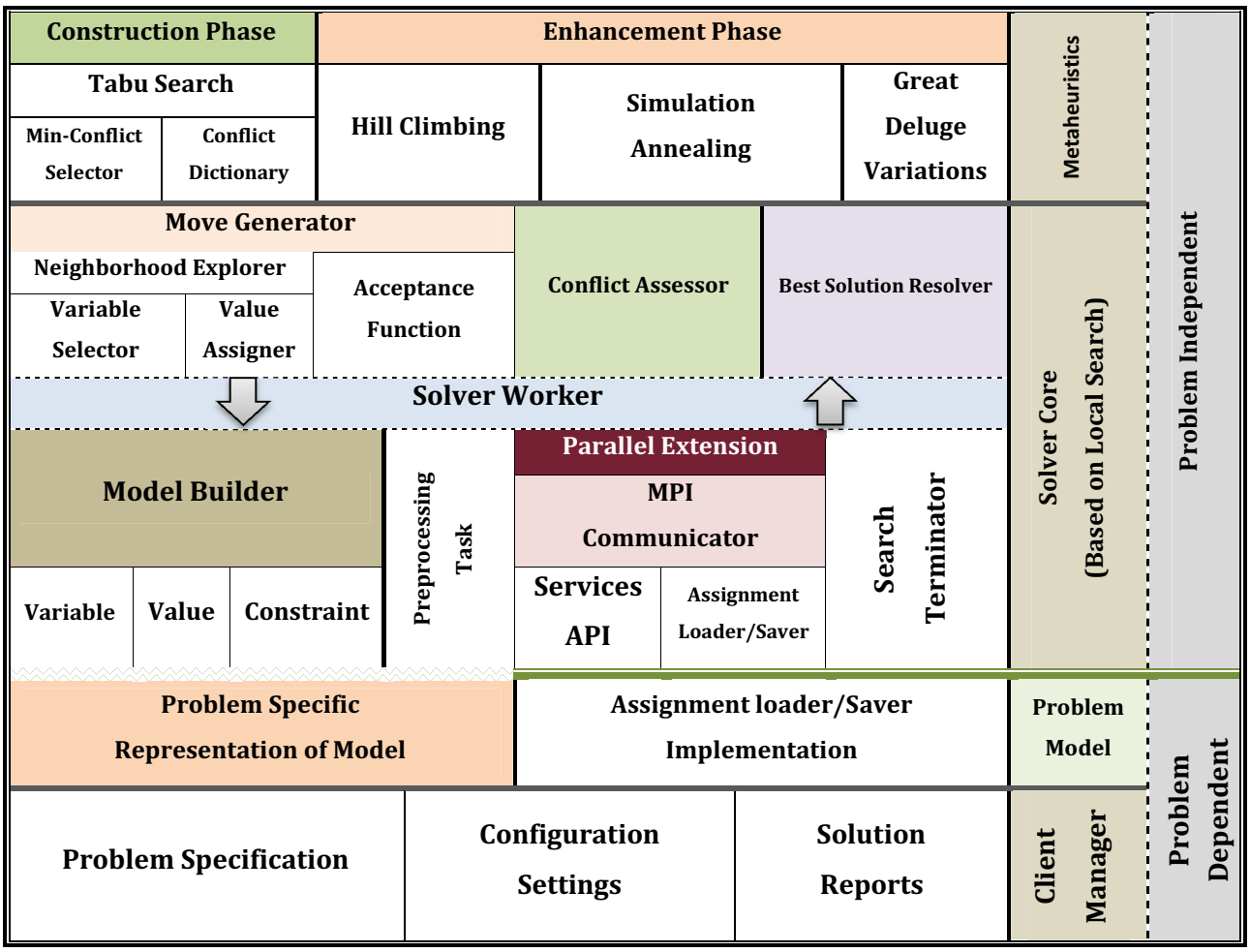


Figure 10 - IDS Solver hierarchy layers

5.2.1.1 Client Manager

Client manager is a problem-specific layer that is responsible for preparing inputs, configuration and reporting outputs. It serves the role of user interface and it is up to the developer to use the appropriate user interface (i.e. console application, forms, or web based interface). It has no role in solving the problem or modelling it

but rather it knows how to communicate with the layer above it, the problem model, to provide problem inputs and configuration settings and receive solutions, assignments and output reports.

5.2.1.2 Problem Model

This layer is also a problem-dependent and part of it, which is concerned with the problem modelling, must derive from the solver core layer. It receives the problem inputs and configuration settings from client manager. It also knows how to load, prepare the model representation for the solver core as well as it is responsible for saving the model at any state. The saving may be as a whole model or just the best solution assignments array. This last functionality is to be used within MPI (message passing interface) stage that is part of Solver Core.

The problem model implements, optionally, a pre-processing stage that can help in adjusting some of the problem inputs and settings to enhance the search task of the solver core. The process of pre-processing may involve discovering hidden constraints as we will see when we introduce solving large examination timetabling problem or computing all variables swaps for moderate sized problems as used in Job Shop Scheduling Problem. The problem model is developed by writing derived core solver's abstract classes that uses the three main CSP aspects (modelled as classes in solver core); variable, value and constraint. Such user-defined classes contain only specific problem description, but no control information on how to achieve the search

algorithm processing as this is the responsibility of the local search stage in Solver Core.

5.2.1.3 Solver core

This is the biggest part of IDS solver. It is considered as the core of IDS Solver and it is composed of a set of cooperating classes that accomplish different aspects of Local Search. Indeed, the relationships between classes, and their interactions by mutual method invocation, are completely dealt with by the framework.

The classes of the framework are split into two main parts separated by solver worker which plays the role of a coordinator between the different classes in solver core. Both parts contain seven categories with different features depending on the role played in a Local Search algorithm. All these categories are internally managed by IDS algorithm.

5.2.1.3.1 Model Builder

The model builder composes of three main base classes; Variable, Value and Constraint. These classes usually can be implemented by the problem dependent part in a way that it should reflect the problem to be solved. These classes have several virtual methods that lays down very basic functionalities for the model.

5.2.1.3.2 Search Terminator

Search terminator class is the class that is responsible for evaluating the criteria for stopping the search process. This by default uses either timeout settings or

maximum iterations or both whichever reached first. If other criteria is used, this has to be overridden by the problem-dependent part.

5.2.1.3.3 Move Generator

Move generator is the part that is responsible for generating a neighbour in local search algorithm and deciding on whether the move is acceptable or not. This component provides only the basic functionality for selecting a variable and a value randomly. Any metaheuristic algorithm can implement and override the basic functionality of the component that fits the requirements of that algorithm.

This can be clearly seen in the different metaheuristics algorithms that we implemented in our solver. For instance, Hill Climbing only accepts only moves that lead to a better solution (whatever better means here). Simulated Annealing, on the other hand accepts moves that leads to a solution value that better that the outcome of a specific function (probability function) provided by the metaheuristic algorithm.

The same rule applies to the acceptance function where the acceptance criteria is different from algorithm to another. By default, the acceptance of a new solution is only based on if it is a better value (i.e. less decimal value). This can be overridden according to the problem model itself by deciding if the optimal solution for the problem is looking for maximising the weight of satisfied constraints (soft and/or hard) or minimising the total cost of violating such constraints.

5.2.1.4 IDS algorithm

As stated earlier, incremental dynamic search is based on ideas from local search. IDS solver operates in iterations where in the course of each iteration, a neighbour is selected. Typically a neighbour consists of a selected variable and assigned value from its domain. This process is known as a neighbour selection, generating a move or neighbour exploration in local search. Variable selection depends on the stage that we are in. If we are during the stage of building complete solution of assignments then normally an unassigned variable is selected. Otherwise if we are optimizing a complete best solution found so far then the selected variable will be an already assigned variable.

This will depend on some other criteria that may relate to violated soft constraints that are typically ignored during the search for complete feasible solution building. Unlike classic local search, we are only looking for feasible solutions. In other words, IDS is not interested in complete infeasible solutions but rather in feasible solutions where no hard constraint is violated whether the solution is complete or partial. For this reason, the selection of a variable and a value assigning might cause some other variables that are involved in conflicting assignment to be unassigned.

This positions IDS in contrast to classic local search, where the search starts with infeasible complete solution where a number of variables might be in contradiction violating many hard constraints.

IDS solver consists of two main phases (as most local search flavours do): Construction and Optimization. In construction phase, the goal is to hand in the next optimization phase a very good quality initial complete (or partial) feasible set of assignments (also known as solution) in the hope that this will help enhancing it in the next phase. In classic local search method, it has been found that it is desirable that the construction phase should continue until 9% of the whole search process has been accomplished, upon which the optimization phase initiates [20].

In IDS, the construction phase should continue till we find the first feasible complete solution or reaches maximum iterations with feasible partial solution. Optimization phase then kicks in and tries, based on model specific heuristics, to enhance the solution found so far.

5.2.2. IDS Architecture

IDS architecture consists of two main parts; IDS core solver and Parallel MPI (Message Passing Interface) component. As illustrated in figure 11. The idea here is to host the IDS core solver component on one or more hosting machines that collaborate with each other to achieve the best solution for the problem at hand using the services managed by Parallel MPI Component.

This Concurrent system consists of distributing several instances of the solver on different hosts to solve a specific task. Each instance, then, may have its own behavior to sequentially solve the problem. The behaviour may be different in the used

algorithms used in the search and/or the parameters used in the different algorithms or in the solver itself. Consequently, this parallel architecture's purpose is to select propagate the best solution to other instances once achieved. The other instances in turn, will have to restart the search process using that best solution.

The key in having several instances using different behaviour to solve the same problem or to have an algorithm which is able to behave differently according to some configurations is that the behavior could be related to some randomization features that work in different parts of the search space and hence this may lead to better results.

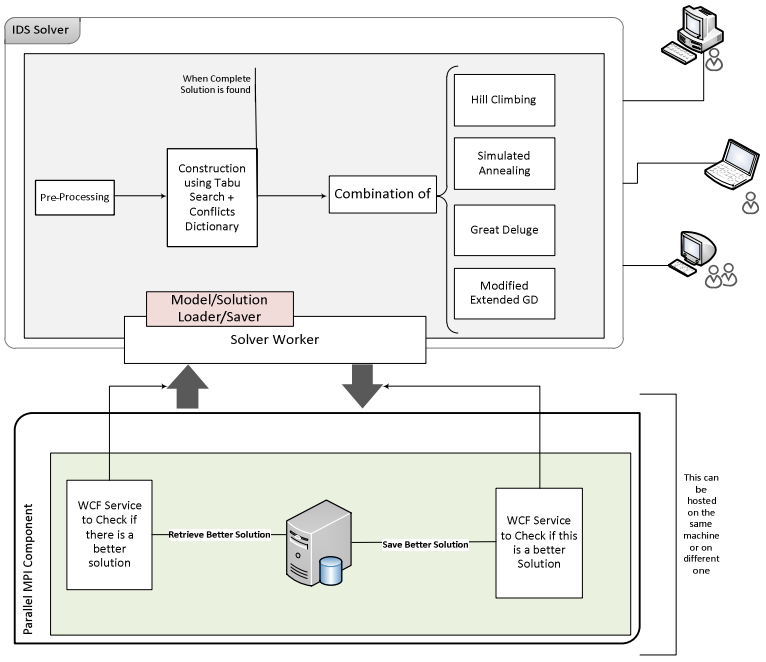


Figure 11 - IDS with Parallel MPI extension architecture

5.2.2.1. Basic IDS algorithm

procedure IDS

Input: *A constraint network* $\mathcal{R} = (V, D, C)$

termination criteria: # *of maximum iterations Or time out*

Cost function $f(s)$ *(Problem dependent)*

Output: *a feasible solution (Complete or partial)*

begin procedure

$\bar{a} \leftarrow$ *initial assignemnts*

$\bar{c} \leftarrow \bar{a} \Rightarrow$ *current assignments*

$\bar{b} \leftarrow \bar{a} \Rightarrow$ *best assignments*

$i \leftarrow 0 \Rightarrow$ *current iteration*

$best_i \leftarrow 0 \Rightarrow$ *best assignments iteration*

while *(termination criteria not met)* ***do***

$i \leftarrow i + 1$

$n \leftarrow$ ***selectNeighbour***$(\bar{c}, i)$

$x \leftarrow$ ***ConflictsAssessor***$(\bar{c}, n) \Rightarrow$ *compute conflicting assignments*

$\bar{c} \leftarrow$ ***setCurrentAssignments***$(\bar{c}, x, n) \Rightarrow$ *Unassign* x *&assign* n

if $($***BestSolutionResolver***$(\bar{c}, \bar{b}, i, best_i) = true)$

$\bar{b} \leftarrow \bar{c}$

$best_i \leftarrow i$

end if

end while

return $\bar{b}$

end procedure

Figure 12 - IDS Algorithm

The procedure *selectNeighbour* is usually implemented using local search heuristics. In IDS solver we have implemented four well-known local search heuristics, tabu search with conflicts dictionary, hill-climbing, simulated annealing and three different flavours of Great Deluge algorithm. Among these three methods, a novel updated version of extended great deluge based algorithm that shall be introduced in Chapter 6 when we solve examination timetabling problem.

Once neighbour is selected, we need to find out how it affects current assignments. And because we cannot have conflicting assignments, all variables that are involved in any conflict with the current neighbour selection must be un-assigned. This is done in $ConflictsAssessor$ procedure.

```
procedure ConflictsAssessor(c̄, n)
begin procedure
x ← nil ⇒ set of conflicts
for each â in c̄
        if â in conflict with n
        x ← x + â
        end if
end for
return x
end procedure
```

Figure 13 - Conflicts Assessor procedure

All returned conflicting assignments then have to be unassigned. The remaining assigned variables will compose the new solution. Based on the current solution, the procedure ***BestSolutionResolver*** checks to see if the current set of assignments is better (using the problem specific cost or penalty function) than the best solution found so far.

procedure BestSolutionResolver$(\bar{c}, \bar{b}, i, best_i)$
begin procedure
$\Delta_b \leftarrow 0 \Rightarrow Best\ solution\ iteration\ delta$
$t_{\bar{b}} \leftarrow f(\bar{b}) \Rightarrow best\ assignments\ total\ cost$
$t_{\bar{c}} \leftarrow f(\bar{c}) \Rightarrow current\ assignments\ total\ cost$
$diff_i \leftarrow i - best_i$
$\Delta_b = calculateBSID(diff_i)$
$if\ (t_{\bar{c}} < t_{\bar{b}} - \Delta_b) return\ true$
$else\ return\ false$
end procedure

Figure 14 - Best Solution Resolver procedure

5.2.2.2. Best Solution Iterations Delta

The procedure ***BestSolutionResolver*** compares the best solution found so far to the current solution that its assignments just have been assigned using a provided cost function. It does have one difference from other ordinary solution resolvers in that it tries to get the best neighbour assignment surrounding the current solution by not to rush to accept the current solution if its iteration is too close to the best

solution's best iteration even if it is a better quality but rather based on a calculated variable called *"Best Solution Iterations Delta"*.

procedure calculateBSID$(diff_i)$ (BSID → Best Solution Iteration Delta)

begin procedure

$\Delta_b \leftarrow 0$

$if(diff_i \geq 1000)\ \Delta_b \leftarrow 0$

$elseif(diff_i \geq 500\ and\ diff_i < 1000)\ \Delta_b \leftarrow 1$

$elseif(diff_i \geq 400\ and\ diff_i < 500)\ \Delta_b \leftarrow 5$

$elseif(diff_i \geq 300\ and\ diff_i < 400)\ \Delta_b \leftarrow 10$

$elseif(diff_i \geq 100\ and\ diff_i < 300)\ \Delta_b \leftarrow 25$

$else\ \Delta_b \leftarrow 100$

$return\ \Delta_b$

end procedure

Figure 15 - Calculate BSID procedure

Best Solution Iteration Delta is calculated based on the how far the current solution, if it is better quality, from the best solution's iteration. The procedure above shows default values but it can be customized using any proper values for the problem at hand. It is worth mentioning that even if a better quality solution found and its iteration is too close, we still save it in memory in case we got far away from the best solution's iteration and we were not able to find a better quality solution. In that case, we use the last saved best solution. This technique can be also named *"solution late acceptance"*.

5.2.2.3. IDS Parallel extension

IDS algorithm can be optionally extended by a parallel extension which in its core works exactly the same as basic IDS but in parallel. IDS Parallel Extension is a simple implementation of the well-known MPI (Message Passing Interface) protocol where two or more instances of IDS solver collaborate in parallel to minimize (or maximize) cost/penalty or weight function through exploring different parts of the search space concurrently.

The MPI [180] in itself is a specification where the goal of it is to provide a widely used standard for writing message passing programs. The interface attempts to be practical, portable, efficient and flexible.

5.2.2.3.1. MPI Component

The MPI parallel component in IDS was designed and implemented using WCF services (Microsoft© Windows Communication Services) to exchange messages and a database engine to save solutions and/or complete problem models typically hosted on a central server. WCF is Microsoft© implementation of distributed computing using Services. WCF is designed using service oriented architecture principles to support distributed computing where services have remote or local consumers. Clients can consume multiple services and services can be consumed by multiple clients. Services typically have a WSDL interface (Web Services Description

Language) that any WCF client can use to consume the service, regardless of which platform the service is hosted on.

MPI component also uses database storage to save problem's best solutions along with a hash code that represents the problem taken from the problem model itself to make sure that when a client communicates with the MPI component, it gets the best solution for the problem at hand. There are two main services in MPI component; one that checks for better solution and another that checks if the current solution is the best so far. IDS instances represented as WCF clients can exchange messages by checking for a better solution found by others or sends a best solution to others. This ideally can be implemented using different clients hosted on different machines and communicate using a central server. But also, all of the infrastructure and IDS instances can be run from within one powerful host machine. It is worth mentioning that MPI services are running on another thread so they do not affect the solver's main thread. Download service only notifies the main thread when there is a better solution ready to be assigned to the solver's main thread. Likewise, upload service, uploads a best solution if it happens to be the best solution found so far and it does that in the background.

- **Save Best Solution Algorithm**

During search, this service is responsible for finding out if the current feasible solution (complete or incomplete) is a better value (by a configurable setting value, default is 25) or more assigned variables solution than the last one saved. If that is

the case, then it converts the solution to represented XML string, compresses it and sends it to database storage. The following algorithm describes its process.

procedure SaveBestSolution
Input: $s \leftarrow current\ Solution$
$pId \leftarrow problem\ Id \Rightarrow usually\ a\ unique\ identifier\ or\ hash\ string$
$n \leftarrow number\ of\ model\ variables$
$a \leftarrow number\ Of\ assigned\ Variables$
begin procedure
$v \leftarrow Value(s) \Rightarrow currentsolution\ value$
$isDuringConstruction \leftarrow a \neq n \Rightarrow boolean\ value$
$bs \leftarrow RetrieveBestSolution(pId) \Rightarrow from\ storage/Data\ base$
$if\ (bs \neq nil)$
$\bar{v} \leftarrow Value(bs) \Rightarrow best\ Solution\ Value$
$\bar{a} \leftarrow AssignedVars(bs) \Rightarrow \#\ Of\ bs's\ assigned\ Variables$
$\bar{d} \leftarrow \bar{a} \neq n \Rightarrow Is\ saved\ solution\ during\ construction;\ boolean$
$if\ (a > \bar{a}\ and\ \bar{d}\ is\ true)$
$SaveBestSolution(pId, s)$
$else\ if\ (not\ \bar{d}\ and\ v < \bar{v})$
$SaveBestSolution(pId, s)$
end Procedure

Figure 16 - Save Best Solution procedure

This is the service that is responsible for checking on the availability of better solution (cost or weight wise). If it finds one, it retrieves it. The best solution in our procedure is saved as compressed XML string. It is compressed so that it would not take much time when downloading or uploading. The XML string represents best solution assignments to problem variables. The service examines the current solution's assignments and whether it is during construction, in other words, searching for the first complete feasible solution. If it is during construction, it looks

for assignments with higher number of assigned variables rather than better value. This turns out to be useful for over-constrained problems where all we look for is more assigned variables rather than best values. If the current solution is complete feasible solution, then it looks for a better value. It is worth mentioning that for performance concerns we have noticed during the development of this service, the calling to this service does not happen in every search iteration but rather on a configured setting value which its default value is 50, so that it leaves some time and space for the current search process to look for better solutions. The next algorithm describes the service's main steps.

procedure GetBestSolution
Input: $s \leftarrow current\ Solution$
$pId \leftarrow problem\ Id \Rightarrow usually\ a\ unique\ identifier\ or\ hash\ string$
$n \leftarrow number\ of\ model\ variables,\ a \leftarrow number\ Of\ assigned\ Variables$
Output: $\bar{s} \leftarrow best\ Solution\ Assignments \Rightarrow if\ available, null\ otherwise$
begin procedure
$v \leftarrow Value(s) \Rightarrow currentsolution\ value$
$isDuringConstruction \leftarrow a \neq n \Rightarrow boolean\ value$
$\bar{s} \leftarrow s \Rightarrow set\ current\ solution\ assignments\ as\ best\ solution\ Assignments$
$bs \leftarrow RetrieveBestSolution(pId) \Rightarrow from\ storage/Data\ base$
$if\ (bs \neq nil)$
$\bar{v} \leftarrow Value(bs) \Rightarrow best\ Solution\ Value$
$\bar{a} \leftarrow AssignedVars(bs) \Rightarrow \#\ Of\ bs's\ assigned\ Variables$
$if\ (a < \bar{a}\ and\ isDuringConstruction)$
$\bar{s} \leftarrow bs$
$else\ if\ (not\ isDuringConstruction\ and\ \bar{v} < v)$
$\bar{s} \leftarrow bs$
$return\ \bar{s}$
end Procedure

Figure 17 - Get Best Solution procedure

5.3 Summary

The fundamental idea behind the development of IDS Solver is to consolidate Local Search techniques and features in one place. The framework provides a model for the design and implementation of Local Search algorithms and reveals numerous advantages concerning implementing the algorithm from start to finish in terms of code reuse, organisation and methodology. Additionally, IDS Solver is a white box piece of software which means it is easily extensible by means of new class inheritance and compositions.

The main goal of IDS Solver is to simplify the task of researchers, implementers and developers. The idea is put in place essential infrastructure core and details of local search algorithms and to leave only the problem-specific development details to the user. However, this is not to downgrade the problem-specific details development. Indeed, in many cases, problem-specific details dominate the implementation of most parts of problem modelling and getting that right is a key factor to getting an efficient problem solving process.

It is certain that generic solvers provide the user a model to lean on, that can easily be derived and implemented and getting that generic part correctly is unarguably equally important as designing the specific part. Using IDS solver or likewise any object oriented frameworks based solvers; will require the developer to conform to

generic solver usage and practice properly any abstract code which leads the user to concentrate more if not fully on the specific details of the problem.

IDS Solver makes use of the best implementation of design patterns and software best practices to gain a better computational efficiency. The system allows the user to optionally pick and mix any local search techniques and to generate and experiment new combinations of features. We tried to implement and use the most known features of local search techniques and methods.

In this chapter, we introduced the main layers and architecture of the solver that will be the basis for the design, model and solving of large examination timetabling in next chapters. We also introduced the message passing interface parallel component that is responsible for communicating best solutions among IDS solver instances collaboration which perform the core of the optional parallel problem solving. Microsoft .NET framework is used as the backbone framework for the solver and as a part of using it; we did a detailed study of the different diverse collections included in .NET. The detailed study along with experiments benchmarking is fully documented in Appendix A.

6. Multi-Phase Parallel Hybrid Metaheuristics Approach to Solve Large ETP

This chapter introduces our proposed search algorithm to solve large ETP. The search algorithm is based on IDS solver algorithm that was introduced in chapter 4. The search algorithm consists of multiple phases. A pre-processing phase, a construction phase and an enhancement phase.

We also propose a novel approach that is used as an alternative approach to the tabu list of moves in Tabu Search that uses a data structure mechanism that represents a conflicts dictionary. An updated version of extended "Great Deluge" is also introduced in details which is used as one of the methods that can be utilized within the enhancement phase of the search algorithm.

6.1 Introduction

The examination timetabling problem is an annual or semi-annual problem for educational institutions. Due to its complexity and practicality, it is extensively studied by researchers in operational research and computational intelligence sectors. Many approaches have been proposed and discussed for solving the problem in the existing literature. These approaches can be divided into the following eight main categories; graph-based, sequential techniques, clustering-based techniques, constraint-based techniques, metaheuristics, hyper-heuristics, multi-criteria

techniques, and case-based reasoning techniques. In chapter 2 we introduced and discussed these approaches. In this chapter we only concentrate on and discuss further the one methodology that we used in our approach; metaheuristics and how we hybridize four of them to perform our search. We also introduce our approach to model the problem as well as the search algorithm pre-process phase used.

6.2 Proposed Algorithm

Our proposed search algorithm is based on IDS solver algorithm that was introduced in chapter 4. The search algorithm consists of three main phases. A pre-processing phase followed by a construction phase and enhancement phase. The pre-processing phase is split into two stages; stage one is the problem's collections ordering process where a sort process takes place for the different collections that the exam problem consists of. These collections are exams, rooms, periods and students. Exams and students are usually large collections and pre-ordering those leads to a better performance and efficient results during search. Stage two of the pre-processing phase is a technique that involves the discovery of unspecified constraints that is used to discover all hard constraints that were not explicitly defined in the problem.

Next phase is the construction phase where a complete feasible solution is found using tabu search metaheuristic along with conflicts dictionary to reduce cycling. Conflicts dictionary is detailed in this chapter and essentially is a dictionary data

structure based consists of a key and value and is used for its performance capability. Each entry in the conflicts dictionary represents a count for the number of conflicts that an assignment causes during search. In future search iterations, the entry with the highest counts are avoided and regarded as a tabu.

Utilizing tabu search metaheuristics with conflicts dictionary can be further detailed as follows. As the search is only considered by variable and value selection criteria, the algorithm initially tries to find those variables that are most problematic to assign. Usually, a variable is randomly selected from unassigned variables that have the smallest domain size and less number of hard constraints. It then attempts to select the best value to assign to the selected variable using conflicts dictionary.

A best value is one that its assignment improves the overall value of the solution. Also, any value that violates a fewer number of hard constraints is considered. In other words, when assigning a value to a variable, the algorithm is looking to minimize the number of conflicting variables that need to be unassigned in order to reach or keep a solution feasible after assignment. A value is selected randomly if there is more than one value with such conditions. Soft constraints violations are totally ignored in this phase as they might affect algorithm performance when searching for complete feasible solution.

In the next phase (enhancement phase), a combination of three metaheuristics are employed and this is up to the user to select just one, two or three out of the three

metaheuristics. This can be configured in the application's file settings. In this phase, whatever a metaheuristics is used, a local optimum is found. Once a solution can no longer be improved or reaches an idle state, another metaheuristic technique kicks in and used. In our algorithm we used three of a well-known metaheuristics. These are "Hill Climbing", "Simulated Annealing" and a variation of Great Deluge including our updated version of extended "Great Deluge" that we will introduce in details later on in this chapter.

The updated extended GD algorithm is altered so that it allows some alternations of the bound that is imposed on the overall solution value. Optionally, the original version of Simulated Annealing (SA) [113] can also be used in any of the stages although results show that updated GD algorithm outperform SA algorithm.

Ideally, we use tabu search with conflicts dictionary in construction phase; Hill climbing or GD in the first phase; and finally updated extended GD or SA in last phase. The search ends after a predetermined time limit has been reached. The best solution found within that limit is returned.

6.3 Problem Modelling

Because exam timetabling problem solving involves a constraint based solver, the problem has to be modelled in a certain way that is understandable to our IDS constraint based solver. Exam timetabling problem is modelled using IDS solver

primitives; variable, value and constraint (soft and hard) where a set of values compose a variable's domain.

IDS imposes (as any constraint based solver) that hard constraints cannot be violated although, during construction phase, this is allowed and variables that are involved in violations or conflicts have to be unassigned. Unlike hard constraints, soft constraints are allowed to be violated and the objective is to minimize the number of violated soft constraints.

Exam timetabling problem is implemented using IDS solver framework as follows;

- **Variable**: Each exam is modeled as a *problem variable.*
- **Value**: Values are represented as exam assignments. Each exam (variable) has a *domain values* consisting of all possible assignments of that exam where assignments are composed of a period and room.
- **Constraint**: In exam timetabling problem, there are several constraints that can be defined.
    - **Room Constraint**: Each exam is subject to be held in certain rooms that have the necessary features and seating capacity so for each room a constraint is created and added to all exams that will be held in that room to prohibit two exams from coincide in the room.
    - **Student Constraint**: For every student, a constraint is created and added on all exams to be attended by that Student. This constraint identifies conflicts where exams attended by the student of the constraint taking place at the same

time. The constraint disallows two such exams to be held at the same time for the same student.

- **Order constraint:** This constraint is about exam ordering and precedence between two or more exams.
- **Same Duration Constraint**: This constraint is about two or more exams that must/should (hard/soft) take place in the same time slot.
- **Different Duration Constraint**: This constraint is about two or more exams that must/should (hard/soft) take place in different time slots.
- **Same Room Constraint**: This constraint is about two or more exams that must/should (hard/soft) take place in the same room.
- **Different Room Constraint:** This constraint is about two or more exams that must/should (hard/soft) take place in different rooms.

To summarize the different resources that the exam timetabling problem contains, we can describe each resource in the problem as follows;

- **Exams:** represented as solver variables.
- **Rooms**: represented as part of composed exam assignments (value), which consists of period and room. Room also can create a constraint.
- **Periods**: represented as part of exam assignment (value), which consists of period and room. Period also can be part of constraints set.
- **Students**: Represented as constraints

6.3.1 Penalty Function

IDS solver contains a generic function that is a problem dependent to calculate the total cost/value of a solution. In exam timetabling problem, the penalty function is implemented by going throw all violated soft and hard constraints and other features that is specific to the problem at hand. Each constraint involves a single or multiple resources and violating it has its own penalty value that should be set in problem description.

In exam timetabling problem, there are a number of penalties that need to be calculated, we mention some of them that are known in most exam timetabling problems as well as others that we will go throw when we introduce our benchmarking data set that we used in our experiment.

1. Two exams in a row
2. Two exam in the same day
3. Mixed durations where two or more exams are taking place in the same room but they have different durations.
4. Mixed durations where two or more exams are taking place in the same room but they have different durations.
5. Room penalty where using certain rooms implies specific penalty to discourage scheduling exam to them.
6. Period penalty where assigning exam to certain periods implies specific penalty.

The total penalty value of any solution is the summation of penalties of all violated soft and hard constraints in the whole exam problem.

6.4 Pre-processing Phase

The difficulty of any exam timetabling depends on three factors; number of students that enroll in it, direct student conflicts in that exam, its scheduling priority constraint, if any, among all exams scheduling. Exams with most scheduling difficulty should be scheduled first. The reason for this is that if scheduled late, they would most likely increase the potential of the un-assignment process of other exams that violate its constraints and hence eventually the process of backtracking. For these reasons, our approach considers a pre-processing phase that enhances the search process. The pre-processing phase consists of two stages; exam timetabling problem collections ordering and the discovery of un-specified hard constraints.

6.4.1 Problem Collections Ordering Stage

In [54], [19] two of the well-known common techniques that describe the ordering of exams based on difficulty criteria preceding their assignment to timeslots. Our approach is slightly different to these techniques. It depends on a different concept revolving around our knowledge that large exam timetabling problems contain large exams, students and resources collections, and enhancing the way that we retrieve and lookup any element in these collections is a key in any efficient search algorithm. Indeed, the time complexity of looking up or retrieving an element from

unsorted large collection is $O(n)$ whereas the time complexity of the same process in a sorted collection is $O(log_n)$.

Our approach of collections ordering pre-processing stage involves the creation of four ordered collections at the time of building problem variables, values and facts collections based on the efficient Microsoft© .NET framework which implements quick sort algorithm. It is worth mentioning that quick sort algorithm makes $O(n\ log_n)$ comparisons in average to sort n items. It takes $O(n^2)$ in worst case scenario. This is due to the fact that a bad choice of partition element could split the array into one element on one side, and the entire rest of the array on the other. If this is the case for every iteration of the process, it takes n iterations to split it down into pieces of one element each. In total, there are n operations, each with a complexity of $O(n)$, for $O(\ n^2)$ overall.

Prior to our decision on whether to perform this stage or not, we thought of two issues; the time needed to build large sorted collections and the time required to lookup or retrieve any element in these collections. As we show in appendix A, based on a small research we did to examine the different types of collections that Microsoft© .NET framework provides, we made the decision to use Sorted Generic Lists for all COP variable, value and constraints collections.

There are two reasons for that. First, we only need to build them once at the beginning of building the problem model and hence we build them in a sorted manner and that

is the only time we spent to sort them. They also do not consume a lot of memory as other types of collections. In fact, they are the least memory consuming collection. The second reason is that after building any of the COP collections, we only need to do lookups which a generic list is good at and one of the fastest collections for that matter and its time complexity is $O(\log_n)$.

The following figure shows the time complexity for adding an element and for looking up or retrieving an element from both unordered and ordered collections. Although, we include the cost of removing an element as in all large collections, as we only build any collection once and lookup or retrieve afterword and there is no need for removals. For the case for collections that needs items removals we use hash sets as the cost of removing an item is considerably small.

	Adding an element	Lookup/Retrieval	Removing an element
Unordered Collection	$O(1)$	$O(n)$	$O(n)$
Ordered Collection	$O(n)$	$O(log_n)$	$O(n)$

Figure 18 - Time Complexity for Ordered and Unordered Collections

6.4.2 Unspecified Constraints Discovery Stage

In any large COP problem that contains a large collection of variables, values and constraints, there is always the possibility of missing some of the hard constraints that depends on some of the declared constraints. Our approach is to provide a pre-processing stage that discovers these unspecified constraints and add them to the

problem's constraints collection. In fact our goal is to add other constraints that should be known before assigning a value to a variable which in essence might eliminate some of the variable's domain values and hence preventing a backtracking process, which would occur later on, if these additional constraints were not specified. It is worth emphasizing that we are not defining any new constraint out of nothing but rather we are discovering new ones based on the already declared ones in problem description to achieve better and efficient search functionality.

Our approach involves building a full graph representation of the exam timetabling problem to achieve discovering possible four types of unspecified constraints. Exam timetabling problem usually contain three types of exam based constraints;

- Exam Ordering: two or more exams scheduling must appear in certain order. For example: $exam_1$ must take place after $exam_2$.
- Exam Coincidence: two or more exams must take place at the same time.
- Exam Exclusion: one or more exams must take place at different times.

This pre-processing stage is based on creating three full graphs for each type of these constraints where the node represents the exam and the edge represents the constraint. If there is an edge between two exams, it means that there is a hard constraint between these two exams. Then by traversing each graph, we try to discover same or different type of constraints between other exams in the same graph. The following are the four steps we use to achieve such discovery.

1. Propagating ordering constraints that belong to concurrent exams: This is about adding new ordering constraints if happens to be one exam or more of concurrent exams has ordering constraint to propagate it to the other concurrent exams. For example: If there is a coincidence constraint declares that $exam_1$ must take place at the same time as $exam_2$ and another ordering constraint states that $exam_2$ must take place after $exam_3$ then a new ordering constraint should be added which states that $exam_1$ must take place after $exam_3$.
2. Propagating ordering constraints that belong to other ordering constraints. For example, if there is an ordering constraint indicates that $exam_1$ must be scheduled after $exam_2$ and $exam_2$ must be scheduled after $exam_3$ then this implies adding a new ordering constraint which states that $exam_1$ must be scheduled after $exam_3$.
3. Propagating coincidence and exclusion constraints that belong to the same exam sets. For example, if there is a coincidence constraint states that $exam_1$ must take place in the same period as $exam_2$ and another distinct constraint states that $exam_2$ must take place in a different period than $exam_3$ then a new exclusion constraint must be added identifies the fact that $exam_1$ and $exam_3$ must take place in different periods.
4. Propagating largest exam period that belong to the same exam set to all other exams if they happen to be involved in the same coincidence constraint. For example, if there is a coincidence constraint that involve three exams, $exam_1$, $exam_2$, and $exam_3$. $exam_1$ has to take place in a period of 3 hours,

$exam_2$ has to take place in a period of $2\frac{1}{2}$ hours and $exam_3$ has to take place in a period of 2 hours. Then all three exams' anticipated periods should be updated to be equal to the largest (3 hours).

Because all three exams belong to coincidence constraint, by updating their proposed period, we avoid a backtracking and un-assignment process if we assign a period of less than three hours to one that indicates that to find out later on that there is another exam that requires a larger period and hence backtracking and value un-assignment. It should be mentioned that any penalty cost that involve periods, when calculated, uses the original period and not the updated period.

6.5 Construction Phase

Local search starts with a construction phase where we look for complete or partial but must be feasible solution. In this phase, we use tabu search metaheuristics local search algorithm. In our approach, tabu search uses IDS algorithm as a local or neighborhood search procedure to iteratively move from one potential solution to an improved one in the neighborhood of the current solution until the stopping criterion has been satisfied. In this algorithm, the search is stopped after either complete feasible solution is found or maximum time is reached (a functionality that is provided and controlled by IDS). The basic concept of Tabu Search as described by Glover [93] is "a metaheuristic superimposed on another heuristic". The overall approach is to avoid cycles by preventing or penalizing moves which take the

solution, in the next iteration, to points in the solution space previously visited and that is why it is called "tabu".

Tabu search metaheuristics is used as neighbour selection (variable and value selection). The most improving assignment or reassignment of a value to a variable is returned pairing in mind that a returned assignment or reassignment can cause un-assignment of other existing assignments. If there is more than one of such assignments, one is selected randomly.

The Tabu search starts with local minima. To avoid retracing the steps used; the method records recent moves in one or more Tabu lists. The original intent of the list was not to prevent a previous move from being repeated, but rather to insure it was not reversed. The Tabu lists are historical in nature and form the Tabu search memory. To avoid cycling, a tabu is maintained during the search. It is the list of the last n selected values. A selection of a value that exists in the tabu list is only allowed when it improves the solution's overall value. The minimum size of the tabu list is a configuration setting called "Tabu List's minimum Size" and maximum size is also a configuration setting named "Tabu list's maximum size". The tabu list starts at minimum size and must be reset to minimum size when a new best solution is found. Tabu list is increased by one each time an idle iterations is reached up to the maximum size. "Idle iterations" is calculated used the following equation.

$$Iter_{Idle} = \frac{Maximum_{Idle}}{List_{Max} - List_{Min}}$$

6.5.1 Conflicts Dictionary

As an alternative approach to the tabu list of moves, we propose a novel approach that uses a data structure mechanism that represents a conflicts dictionary. In this approach, instead of recording recent moves, we record moves with an accumulated count of conflicts that caused. The idea behind conflict dictionary is to work as memory storage for previous clashes to avoid their potential reappearance in future. For example, when a value v_i is assigned to variable V_i , this assignment might cause some hard constraints to be violated and hence the un-assignments of some values from their variables prior to assigning value v_i to variable V_i.

The number of un-assignments (number of conflicts) along with the assignment is added to the conflicts dictionary. The move (assigning value v_i to variable V_i) represents the dictionary entry's key and the number of conflicts represents the dictionary entry's value. If the move is already exists in dictionary we add another entry to the dictionary with the new number of conflicts. In addition to the conflicts dictionary essential functionality, we applied an iteration distance mechanism that records at which iteration an assignment move along with its number of conflicts occurred.

Then, during later search iteration, if the variable is selected again for assignment, the stored information in conflicts dictionary helps guiding the decision on which value should be assigned to the variable. It is actually tells us how many accumulated

potential conflicts on each of the values that stored with it in each move. In other words, all moves that involve this variable will be retrieved from conflicts dictionary and a min-conflict value selection heuristics is applied, which selects the entry with the least number of accumulated conflicts and the dictionary entry key value is assigned to that variable.

Because we use iteration distance mechanism, based on a configuration setting called "Iteration Distance" we exclude any entry that its iteration difference to current iteration is greater than "Iteration distance". If iteration distance is zero, the whole iteration distance mechanism is ignored. For instance, let us suppose that conflict dictionary entries for some move are as follows;

$$\overbrace{00091}^{Dictionary\ entry} \quad \underbrace{(V_{10}, v_{25})}_{Assignment\ (move)} \Longrightarrow \underbrace{8}_{\#\ of\ conflicts}, \underbrace{340}_{Iteration}$$

…

$$\overbrace{02120}^{Dictionary\ entry} \quad \underbrace{(V_{10}, v_{25})}_{Assignment\ (move)} \Longrightarrow \underbrace{4}_{\#\ of\ conflicts}, \underbrace{12983}_{Iteration}$$

…

$$\overbrace{11343}^{Dictionary\ entry} \quad \underbrace{(V_{10}, v_{25})}_{Assignment\ (move)} \Longrightarrow \underbrace{5}_{\#\ of\ conflicts}, \underbrace{34531}_{Iteration}$$

…

$$\overbrace{15992}^{Dictionary\ entry} \quad \underbrace{(V_{10}, v_{25})}_{Assignment\ (move)} \Longrightarrow \underbrace{2}_{\#\ of\ conflicts}, \underbrace{40568}_{Iteration}$$

Assume also that the current iteration is *54390* and the "Iteration distance" is *20000*, then we exclude the first two entries and we only consider the last two as the only ones that their iterations is less than *20000* in iteration difference from the current iteration. And hence, the accumulated conflicts count is 7.

It is worth mentioning that the conflicts dictionary's data structure is represented in our approach by a hash set. Hash set is one of the data structures that Microsoft© .NET framework provides and is based on hash table and we show (in appendix A) that they perform very well when used as key-value data set. Hash sets also has the distinguished feature, as any set, of adding more than one entry with the same key which turned out to be beneficial in our iteration distance mechanism.

6.6 Enhancement Phase – Composing Metaheuristics Search

After a complete feasible solution is found a configurable enhancement search phase starts. It is worth indicating that enhancement phase only commences if a complete feasible solution is found. It is composed of one or more of the three metaheuristics search techniques we used; Hill Climbing Simulated Annealing and Modified Extended Great Deluge. The modified extended great deluge is introduced in new search algorithm that is based on extended great deluge that is described in this chapter and used in an attempt to get better and competitive results. The main functionality of this phase is to enhance and improve current feasible solution (usually complete) that was obtained in previous phase.

6.6.1 Hill Climbing Metaheuristics Search

After a complete feasible solution is found, a basic Hill Climbing algorithm [30], [36], [167], [128], [111] can be used, but not necessarily imposed, with the aim of finding a local optimum for the next phase to start optimizing further the overall solution value. In the course of each iteration, an alteration to an existing assignment is proposed by random selection from solution neighborhood. The generated move is only recognized and accepted when it improves the overall solution value usually when it decreases the number of violated soft constraints not to mention that only changes that do not violate any hard constraints are considered.

The hill climbing phase is complete when it reaches a value named "Hill Climbing Idle Iterations" in which a solution is no longer improved for a certain number of iterations. "Hill Climbing Idle Iterations" is a configuration setting that can be defined or defaulted to a value of *15000* within the solver application if it was not configured.

6.6.2 Simulated Annealing Metaheuristics Search

In our approach, Simulated Annealing metaheuristics search [113] is used optionally based on configuration setting. The control is passed to it either from a previous enhancement search method based on reaching maximum iterations or when no improvement has been occurred for a specific number of iterations. Control can also be passed directly from construction phase.

Simulated Annealing search method, as most stochastic search techniques, is used when the structure of search space is not well understood or is not smooth. Simulated annealing is a minimization technique which has proven to give good results in avoiding local minima; it is based on the idea of taking a random walk through the space at successively lower temperatures. The goal is to find a neighbour in the search space at which cost function or overall solution value (*energy function* in SA) is minimized.

In our attempt in enhancement phase, SA is implemented using a temperature parameter named T. A generated neighbor assignment is accepted if it does not degrade the overall value of the current solution, or with the probability of taking a step is given by a Boltzmann distribution [118].

$$p_{accept} = e^{\frac{-(E_{i+1}-E_i)}{kT}}$$

In other words, a step will occur if the new energy (overall current solution value) is lower. If the new energy is higher, the movement acceptance can still occur, and its probability is proportional to the temperature T and inversely proportional to the energy difference $E_{i+1} - E_i$ ($p_{accept} > 1$). Where E_{i+1} represents the overall value of the current solution and E_i represents the overall value of the last solution.

The temperature T is initially set to a high value and is set by a configuration settings named "*Initial Temperature*" and defaulted to "*1.0*" and a random walk is performed at that temperature. If the movement was not accepted, the temperature is lowered

according to a cooling rate. Cooling rate value is also set by a configuration setting, defaulted to 0.90, and to cool the current temperature down we multiply the current temperature by this value.

If there was no enhancement in solution's overall value after I_x iterations, the best ever solution found is re-established. The value of I_x is calculated according to this formula.

$$I_x = R_{lc} * T_l$$

Where R_{lc} is a configuration setting represents reheat length coefficient and defaulted to 4 and T_l is a configuration setting that represents temperature length and defaulted to 15000.

The search ends when a maximum iteration is reached and handed over to another enhancement metaheuristics search method (if any configured).

6.6.3 Modified Extended Great Deluge Metaheuristics Search

As Hill Climbing and Simulated Annealing metaheuristics search methods, the control is passed to the "Modified Extended Great Deluge" either from a construction phase or from another enhancement search method (HC or SA). Our approach is based on the Extended Great Deluge (EGD) [125] method which in turn is based on the original Great Deluge.

Great Deluge algorithm was introduced by Dueck [84] as a cure to Simulated Annealing's requirement to find a cooling schedule for a particular instance of a

problem (Problem Dependent). GD algorithm starts with a "water level" equal to initial solution value, and a preconfigured rate usually named "tolerance rate" to decrease that water level. The predetermined rate is the only parameter for this algorithm and this is one of this algorithm's advantages.

The algorithm accepts worsening solutions if the penalty cost is less than the water level. The water level is decreased by the pre-determined rate set for every iteration. When Dueck introduced algorithm, he applied this algorithm to solve the travelling salesman problem. The level was decreased by at least 0.01 or by the difference between the level and the length of the current tour divided by 500. Due to the advantage of using less parameter, GD algorithm has been used in several other implementations of metaheuristics.

Burke and Newall [25] apply it to solve examination timetabling problem and produced good results on 13 benchmark datasets (Toronto benchmarking datasets). They used another name for the preconfigured rate to decrease the water level as "decay rate". In their research, the performance of simulated annealing, hill climbing is compared to the great deluge algorithm.

The decay rate is computed as the initial solution quality multiplied by a user provided factor divided by the number of iterations. 20,000,000 and 200,000,000 iterations were used in the algorithm and terminated only if there is no improvement in the last 1,000,000 iterations. The great deluge algorithm's results showed that its

performance is better when compared to simulated annealing and hill climbing. It also performs way better in most cases compared to other methods such as tabu search method and graph colouring metaheuristics.

The Extended Great Deluge (EGD) [125] has a construction phase followed by improvement phase. Construction phase is applied using the existing adaptive ordering heuristic search method [25]. The construction phase uses a weighted order list of the examinations which is to be scheduled based on soft constraints and a constraints named "difficulty to schedule". Once an exam is scheduled, it weight is increased based on localized penalties it came across. The unscheduled examinations are given a considerably larger increase, based on a formulation that is based on the maximum general penalty encountered from [25].

The improvement phase starts when feasibility is achieved in construction phase and it tries to provide an improved solution. Unlike EGD, our approach is only concerned with the enhancement phase and it only tries to improve the overall value of the current feasible complete solution. We use EGD as basis for our algorithm; although our approach is slightly differs as we will see in the next section.

```
Set the initial solution "s" using a construction heuristic
Calculate initial cost function f(s)
Set Initial Boundary Level B = f(s)
Set initial decay Rate B* based on Cooling Parameter
While (stopping criteria not met do)
        Apply neighbourhood Heuristic s* on s
        Calculate f(S*)
        if f(s*) ≤ f(s) or f(s*) <= B Then
            Accept s = s*
      endif
        Lower Boundary B = B – B*
        if no improvement in given time T Then
                Reset Boundary Level B = f(s)
                Set new decay rate B* based on Secondary Cooling Parameter
        endif
endwhile
```

Figure 19 - EGD Algorithm

In EGD algorithm, shown in figure 19, the decision to set a new decay tolerance value is taken based on a specific passed time to test and see if the solution has not been improved. Also, new solution value is accepted not only based on the current solution quality if it is less than water level but rather on the current solution is less than water level or less than best solution quality.

Our approach is different from EGD in two points:

1. In original GD, the tolerance value starts with the initial solution's value and decreased by a preconfigured rate. It tries to range within all neighbours of the current solution in each iteration. However, in our approach, tolerance rate ranges between values that are percentage of the current solution value; one above and one below. In our approach, we use two preconfigured values, namely tolerance

lower bound and tolerance upper bound. Tolerance upper bound is a preconfigured value that defaults to $(108\%)^{iter_{idle}}$ of the initial solution. $iter_{idle}$ is a counter that starts with 1 and incremented by 1 each time tolerance rate is reset. Tolerance lower bound is also a preconfigured value that defaults to 92% of the initial solution. The tolerance decay rate is a predetermined rate that defaults to 99.99995%. At the beginning a tolerance rate t is assigned to a value of the initial solution. It is decreased by tolerance decay rate in each iteration. Likewise, in every iteration, a new neighbour is selected and tested against the current t and the best solution value. If it is better than either one of them, the current solution becomes the best solution and t is decayed by tolerance rate.

2. The second difference occurs at the time of taking the decision to reset the tolerance value *t.* Tolerance value t is reset based on two factors; either t reaches the tolerance lower bound (which as we discussed is equal to 92% (or predetermined value) of the best solution so far. The second choice is to reset it based on the last n (defaulted to 40) solutions if they happen to be consistent and carry the same value, this means that we are stuck in a local optimum and there is no need to complete the full cycle and reach the lower bound. Rather, we decrease the current tolerance decay rate by half the rate and restart.

The following figure show the full algorithm for *"Modified Extended Great Deluge"*

Procedure Modified Extended Great Deluge (***MEGD***)

Input: $s \leftarrow initial\ solution \Rightarrow Complete\ feasible\ solution$

$termination\ criteria$: $\#\ of\ maximum\ iterations\ Or\ time\ out$

$f(s) \leftarrow Cost\ function\ f(s)$

$t^* \leftarrow Decay\ Rate, should\ be < 100\%\ , default = 99.99995\%$

$t_{upper} \leftarrow Upper\ tolerance\ bound\ Rate, default = 108\%$

$t_{lower} \leftarrow Lower\ tolerance\ bound\ Rate, default = 92\%$

$\bar{n} \leftarrow number\ of\ last\ solutions\ values, default = 40$

Output: $an\ enahnced\ complete\ feasible\ solution$

Begin Procedure

$t \leftarrow f(s) \Rightarrow initial\ tolerance\ boundary\ level$

$f \leftarrow f(s) \Rightarrow initial\ solution\ value,\ iter_{idle} \leftarrow 0 \Rightarrow number\ of\ idle\ iteraions$

$sol_{array}[\bar{n}] \leftarrow 0 \Rightarrow arrayof\ last\ \bar{n}\ solutions, n \leftarrow 0$

while $(termination\ criteria\ not\ met)$ ***do***

$s^* \leftarrow$ ***selectNeighbour***(s, i)

$f^* \leftarrow f(s^*) \Rightarrow calculate\ \ current\ solution\ value$

if $(f^* \leq f\ or\ f^* \leq t)$

$s \leftarrow s^* \Rightarrow Accept$

$sol_{array}[n] \leftarrow s^*$

$n \leftarrow n + 1$

$f \leftarrow f^* \Rightarrow current\ solution\ value$

endif

$t \leftarrow t \times t^* \Rightarrow decrease\ boundary$

if$(sol_{array}\ have\ \bar{n}\ values\ and\ all\ are\ the\ same) \Rightarrow Stuck\ in\ local\ optimum$

$iter_{idle} \leftarrow iter_{idle} + 0.5 \Rightarrow increase\ idle\ iterations\ by\ half$

$t \leftarrow \left(t_{upper}\right)^{iter_{idle}} \times s \Rightarrow New\ Tolerance\ Boundary\ \ Level$

$sol_{array}[\bar{n}] \leftarrow 0 \Rightarrow reset\ last\ solutions\ values\ array$

$n \leftarrow 0$

else

$t_{level} \leftarrow (t_{lower})^{iter_{idle}+1} \times s \Rightarrow Tolerance\ Boundary\ \ Level$

if $(t \leq t_{level})$

$iter_{idle} \leftarrow iter_{idle} + 1 \Rightarrow increase\ idle\ iterations$

$t \leftarrow \left(t_{upper}\right)^{iter_{idle}} \times s \Rightarrow New\ Tolerance\ Boundary\ \ Level$

endif

endif

end while

return s

end procedure

Figure 20 - MEGD Algorithm

6.7 The Parallel Approach

Our parallel approach is backed by MPI technique that is managed by the IDS algorithm, even though solving exam timetabling problem makes no difference from solving any other problems. As we discussed in chapter 4, the Message Passing Interface (MPI) parallel technique relies on message passing using web services (Microsoft© Windows Communication Services "WCF" services) from one instance to another via central server and central database. The parallel approach involves running two or more instances of the same solver to do the search for the same problem. These instances can run on different machines or on the same machines, although the latter requires a powerful machine to get best results in relatively fast times.

Running solver in parallel is based on a configuration setting that indicates that. If running in parallel, the solver tries to retrieve the best solution that is saved in server database for that problem instance. The solver, then, loads up that solution including number of iterations and time and at that point the search resumes. When the solver finds a better solution quality, the solver attempts to save that solution along with its iteration and time (namely best iteration and best time) to be used by other instances or in futures searches.

It is worth noting that although the parallel approach is a generic one that can be implemented once and run to help solving any problem model, the "save" and "load" functionality that compose the basis for the parallel methodology are problem-

specific. In exam timetabling problem, the best solution is saved in only one of two ways; either a full representational of problem description including assignments or only the best solution assignments.

Loading best solution method in exam timetabling loads all problem assets. In the case where only solution assignments are used, the loader uses original problem model to load up all other variables and values and build related constraints. If the loader uses the whole model representation, nothing else is needed to be loaded as it can build up the whole problem graph from the saved model.

6.8 Neighbourhood Selection

Neighbourhood selection variation is by far the most influential technique that affects rapid local search. Using more than one neighbourhood within a search provides a very effective technique of escaping from a local optimum. It is notable that if the current solution is in a local optimum in one neighbourhood, it might escape the local optimum, if assigned a different neighbourhood and can consequently be more improved using a good feasible approach.

In exam timetabling, the neighbourhoods used in local search techniques largely involve moving some exams from their current timeslot and/or rooms to a new timeslot and/or rooms. Based on that, our implementation uses the following seven neighbourhoods:

1. **Exam Duration Move:** Selects a single exam randomly and move it to a different feasible time slot randomly.
2. **Exam Duration Swap Move:** Selects two exams randomly and swaps their assigned time slots.
3. **Non Conflicting Assignment Move:** Selects an exam randomly and assigns it to a non-conflicting assignment (time slot and room) randomly.
4. **Room Move:** Selects a single exam randomly and move it a different feasible room randomly
5. **Room Swap Move:** Selects two exams randomly and swaps their assigned rooms.
6. **Exam Swap Move:** Selects two exams randomly and swaps their assignments (i.e. time slots and rooms).
7. **Random Move:** Selects an exam randomly and assigns a new assignment to it randomly. The assignment consists of a room and time slot and might cause conflicts.

6.9 Summary

This chapter has presented our proposed approach to solve exam timetabling problem using hybrid metaheuristics that is optionally can be run in parallel using MPI functionality utilizing different instances hosted on one or different machines. It employs four different metaheuristics search methods; tabu search, hill climbing,

simulated annealing, and different flavours of great deluge. We also introduced a proposal for a pre-processing phase that its aim is to enhance the overall search process. A tabu metaheuristic search method with conflict dictionary is introduced as a construction phase to achieve a partial or complete initial feasible solution. The tabu list does not contain operators or moves that are problem specific. It only needs to store the conflicted moves along with accumulated number of conflicts it caused. A modified extended great deluge heuristic search method is detailed that is used during search which should eliminate some of the time wasted in local optimum based on certain conditions.

The selected heuristics process in sequence to produce a good solution for the current state of the problem. The whole hybrid heuristics approach is configurable and able to manage and control its heuristics without having a domain pre-knowledge of the exam timetabling problem.

7. Case Study: Experiments, Analysis and Evaluation

This chapter presents numerical proof that through a comprehensive full testing process over a well-known benchmarking datasets of ITC 2007, our proposed approach is successful in competing with benchmarking results published in literature so far.

Section 7.1 goes through our experiments set-up and resources. In addition, this section also presents the two main themes that are involved in our experiments. In section 7.2, we measure the general behaviour and performance of our implementation in the two different phases to solve exam timetabling problem; construction phase and enhancement phases. Section 7.3, presents full testing results and analysis for our approach on the different benchmarking datasets in both testing themes; sequential and parallel. We also present graphs and tables to compare the different variations of our metaheuristics methods with each other and with ITC 2007 exam track winners.

7.1 Testing Environment

Our testing phase consists of two main themes. The first theme is to test sequentially on a single machine. This is in accordance with ITC 2007 rules which states that "*Participants have to implement an algorithm to tackle the problem on a single processor machine; they can use any programming language.*" [62]. Results out from this category are compared to the results from ITC 2007 Exam Track. It is worth

mentioning that in this theme, we tested all of our variations that we described in chapter 6.

The second theme is to test our proposed parallel approach using three machines. One of these machines will be used as a server that will host a database and provide services for uploading and downloading best solutions. The other two machines will be used as clients that would run two isolated instances of the solver and collaborate to solve the same problem (data set) through means of communication using WCF services hosted on server. Results out of this theme with all variations will be compared to our sequential results in terms of best solution values and best average values.

We performed our experiments on three machines which their specifications are as follows:

- Server machine has AMD Athlon® 64 X2 Dual Core Processor 4200+ 2.20 GHz with 3 GB Memory. This machine is used only in the second category testing (parallel testing) as a hosting server.
- Two clients machines, both are Intel processor based machines
    - One is Intel Core 2-Duo 2.4 GHz processor with 8 GB memory
    - Second is Intel i7 1.6 Quad Core Processor with 8 GB memory.

ITC 2007 organizers provided a benchmarking program that is designed to test approximately how fast your machine is at running timetabling application. The

program will let you know how long you can run your algorithm for, on each of the competition timetabling problem instances. ITC 2007 organizers also stated that the benchmark application is only suitable for single processor machines on their own and not suitable for specialist parallel machines or clusters. We ran this application on the two client machines used in our testing and we found out that the Intel Core 2-Duo processor machine can run for a maximum of 377 seconds and the Intel i7 processor machine can run for a maximum of 362 seconds. So we set our configurations files on both machines to go with these timeout values prior to both testing themes, sequential and parallel.

7.2 Experiment Design and Testing

We used our case study of ITC 2007 exam track as a basis for our experiment. As stated above, testing phase consists of two different themes. One is to run sequentially on one machine and the other is to run in parallel on two machines managed by another server machine. Both themes involve pre-processing stage that consists of ordering problem collections and discovering unspecified constraints. All runs' full history has been logged for deep analysis. Below, we describe the details of the experiments and the order in which we carried out these experiments.

7.2.1 Testing Methods Variations

IDS Solver consists of two phases, construction phase which is based on Tabu search with conflicts dictionary and enhancement phase that consists of multiple

algorithms that can be mixed and combined optionally based on configurations settings. Enhancement phase's algorithms are Hill Climbing, Simulated Annealing, Great Deluge variations (Standard Great Deluge and Modified Extended Great Deluge). In enhancement phase, a neighbour is randomly generated from a set of problem specific neighborhoods and assigned to an exam. This is the case in the hill climbing, simulated annealing and the different versions of great deluge algorithms. And because we are only improving the already constructed complete solution, only moves that do not violate any hard constraints are considered.

	Preprocessing	**Construction**	**Enhancement**
Method 1	No	TS with CD	HC + SA
Method 2	Yes	TS with CD	SA only
Method 3	Yes	TS with CD	EGD only
Method 4	Yes	TS with CD	MEGD only

Table 6 - Testing Methods with used algorithms

Among all combinations of the different algorithms variations, we decided to limit our testing to only 4 variations that are detailed above in table 6 as they reflect what we need to test. These 4 variations will be used in both sequential and parallel testing themes.

In our experiments we set our goal to test the different parts of the system: the whole framework, both construction and enhancement phases which include all of

our new and modified algorithms, preprocessing phase, Tabu Search with conflicts dictionary and modified extended great deluge.

7.2.2 Algorithms Parameters Value Settings

An experiment is always distinguished by different features. These features can be summarized as a problem type, a parameter setting, a set of instances, the number of independent runs of the algorithm on each instance, and a set of problem-dependent performance characteristics. The different algorithms used in our experiments have different parameters that can be set with the help of a configuration file. Based on a preliminary study that involved a pre-test runs, but not to the extent of fine tuning, using the 12 datasets, the parameter values were set manually. The idea is to detect what are the best algorithms parameters values that will result in considerably best solution values. This is also in accordance with ITC 2007 rules where all configuration settings must remain the same during the whole 11 trials for the 12 datasets. These parameters remained the same during the life cycle of the testing phase in the sequential and parallel themes. Table 7 shows parameters for each algorithm that were used in our testing experiments. It is worth noting that SA temperature reheat rate is calculated using the following equation:

$$SA\ temp\ reheat = \left(\frac{1}{Cooling\ Rate}\right)^{Reheat\ Length\ Coefficient \times 1.3}$$

Parameter	Setting Value
TS Iteration Distance	20,000
HC idle iterations	30,000
SA Initial Temperature	150%
SA Temperature Length	23,000
SA Cooling Rate	95%
SA Temperature Reheat Length Coefficient	4
SA Temperature Reheat Rate	131%
GD, EGD and MEGD Lower Bound Rate	90%
GD, EGD and MEGD Upper Bound Rate	112%
GD, EGD and MEGD Cooling Rate	99.99995%

Table 7 - Algorithms Parameters Settings Values

7.2.3 Pre-processing phase

The main purpose of the pre-processing is to try to achieve a good starting performance that could be supportive during the search's two phases; construction and enhancement. It runs before commencing the search process to perform two main processes. The first is to order problems' collections so that retrieving or accessing collections items would be faster and the second is to discover unspecified constraints on the twelve exam track datasets.

Discovering constraints that were not stated in problem description would be beneficial for variables that share one or more constraints as it will reduce the number of times where a backtracking would happen as a result of violating unspecified constraints. Table 8 shows total and detailed numbers of the unspecified

constraints revealed during this stage. Dataset 3 and 11 have the most unspecified constraint while dataset 12 has no undiscovered or modified constraints.

Instance #	Extending Period	Ordering	Coincidence/Exclusion	Total
Instance 1	1	1	0	2
Instance 2	3	0	3	6
Instance 3	19	0	0	19
Instance 4	0	0	0	0
Instance 5	6	1	4	11
Instance 6	13	1	2	16
Instance 7	6	0	0	6
Instance 8	2	1	0	3
Instance 9	1	1	0	2
Instance 10	8	3	2	13
Instance 11	19	0	0	19
Instance 12	0	0	0	0

Table 8 - Solver Pre-processing Stage's Added Constraints

7.2.4 Solver Experiments

Our experiment and testing stage examines the solver's abilities to find the best known solutions for the benchmarking datasets. We measure the performances of the implementations in two different phases. First we only measure the performance of construction phase and we randomly selected 3 of the benchmarking datasets to experiment with. Then, we turn our attention to enhancement phase again with the same experiments and we also select only three datasets. The data collected in both experiments are presented in the following two sections.

7.2.4.1 Construction Phase Testing and Analysis

Our construction approach is based on Tabu Search with Dictionary Conflicts, we set our goal to get a complete feasible solution as fast as possible so that the enhancement phase can kick in and improve the overall solution value gradually. In order to measure the performance of Tabu with CD, we tested it against standard Tabu search and in both cases preprocessing phase is done prior to constructing complete solution.

As known, standard TS algorithm prevents cycling by using the so-called tabu list, which determines the forbidden moves. This list stores the most recently accepted moves. The inverses of the moves in the list are forbidden. The list in standard tabu search in usually limited in size and that is why it is implemented as queue. Our approach differs in that we sum all the accumulated number of conflicts that a move caused rather than just moves which are considered as forbidden. We also implemented "Iteration Distance" which excludes entries that are far away from current iteration based on configured setting for iteration distance.

For the purpose of the construction phase testing, we selected only three datasets out of the twelve datasets, dataset 1, 4, and 5. Dataset 1 is has a moderate conflict density (5.04%) with a high number of students of around 8000 and about 600 exams. Dataset 4 has got a high conflict density (14.94%) along with high number of students and exams which makes it as one of the toughest problem to solve in our

benchmarking datasets. Dataset 5 has the least conflict density among all other datasets with only 0.87% but with higher number of students and exams than dataset.

During the process of building a complete feasible solution, we record solution value in every iteration along with its time. This testing is only concerned with the construction phase and so we set our testing to run for 10 times for each method on every dataset of the selected datasets. Then we select the trial with the best solution value from the ten trials. We represent each point in the graph with the corresponding penalty cost monitored after every iteration along with its time. The last penalty cost is the cost of the first complete feasible solution and that is where the construction phases stops.

It is worth noting that although construction phase might lead to a relatively worse complete solution that might not be the case when eventually it is handed to enhancement phase that might or might not lead to a best solution. In most times, it depends on getting a good quality complete solution which because local search is a stochastic search; it is hard to tell if an initial complete feasible solution is of a good quality or not. Nevertheless, in our approach, getting a complete feasible solution in a fast time is our goal.

Figure 21 illustrates the full snapshot of the best trial for standard TS on dataset 1 while figure 22 shows the same pattern for TS with CD. Among 10 trials, using best run's solution value, although standard TS shows better complete solution (6041), it took 8.03 seconds and 1081 iterations to get it while TS with CD with 6803, took 4.21

and 672 iterations. Also, standard TS algorithm shows relatively higher number of fluctuations between lower penalty cost and higher ones where TS with CD seems to have gradually been building the complete solution with a minimum number of instability.

However, dataset 4 has shown a different pattern. Dataset 4 is one of the most constrained problems; figure 23 articulates how standard TS struggled with finding the less penalty cost solutions in contrary to TS with CD (figure 24). Standard TS spent a total of 28.54 seconds (3281 iterations) to find a best complete solution, amongst 10 trials, with a penalty cost of 31133 while TS with CD took only 2.03 seconds (567 iterations) to find one with 27633. That is a performance improvement of around 1400% with solution value improvement of 112%.

Dataset 5 is the least constrained problem with only 0.87% but with relatively high number of students and exams (9253 students and 1018 exams) which leads us to think that it should be one of the easiest problems to solve. We can see that in the lack of any fluctuations between worse and better solutions in the graphs for the dataset in figures 25 and 26. Nonetheless, TS with CD algorithm performs slightly better than standard TS even though the problem itself tends to be easy to solve. In ten trials, standard TS obtained 6530, as a best solution value, in 2.58 seconds (1020 iterations) while TS with CD achieved 5030, as a best solution value, in 2.21 seconds (1050 iterations).

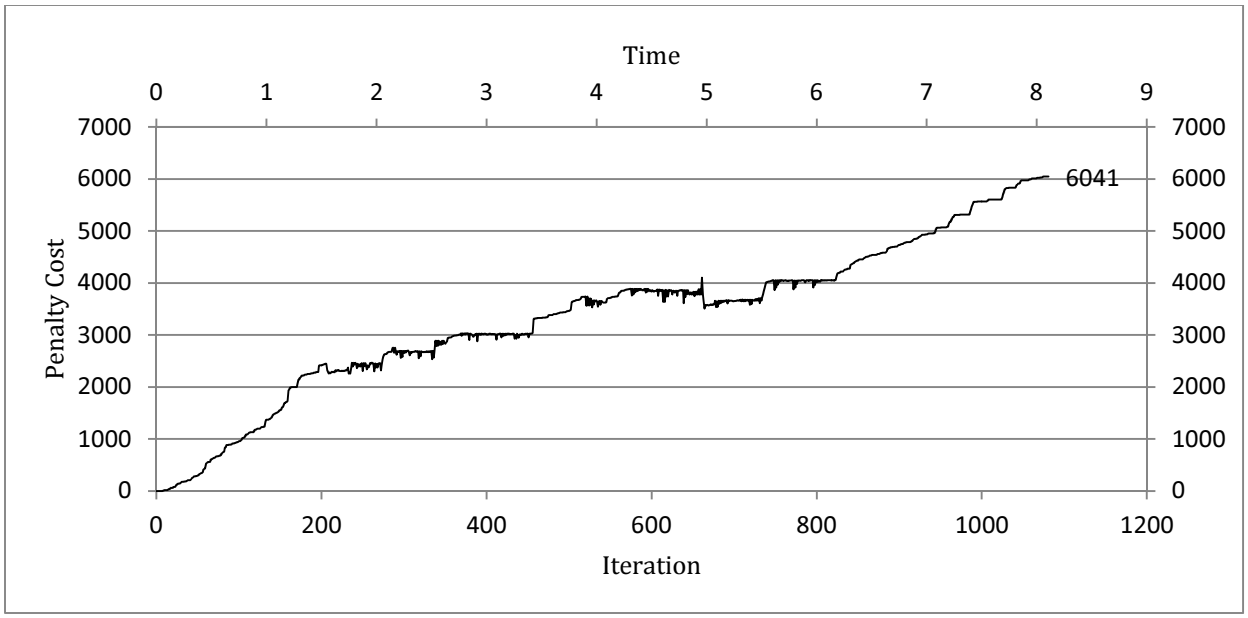


Figure 21 - Dataset 1 Construction Phase using Standard TS

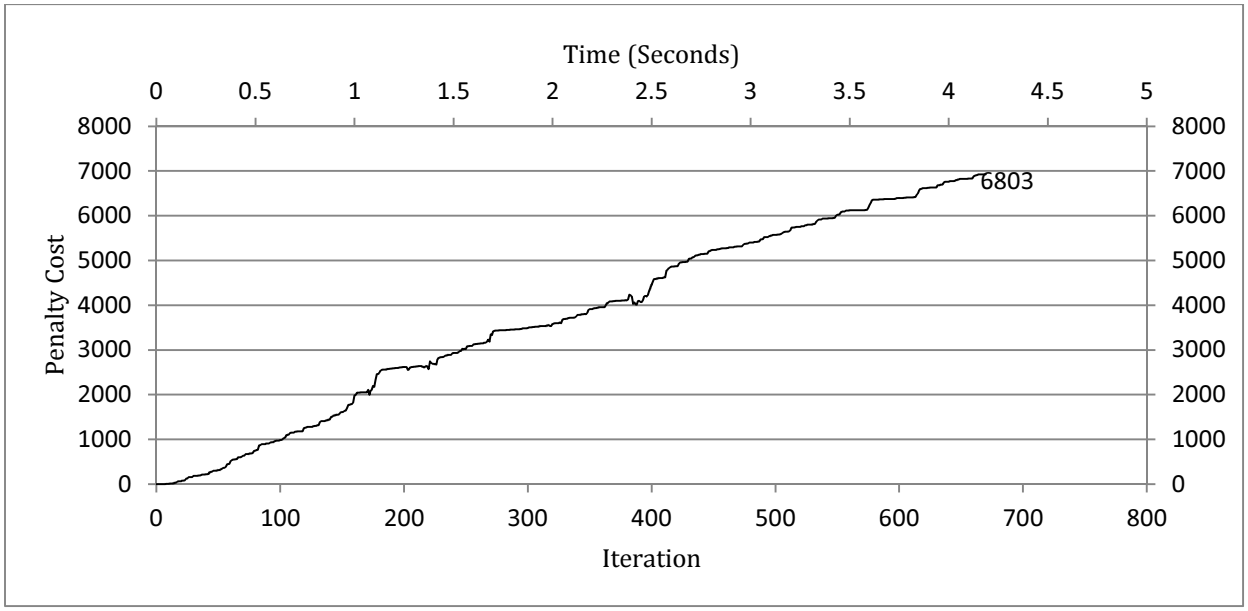


Figure 22 - Dataset 1 Construction Phase using TS with CD

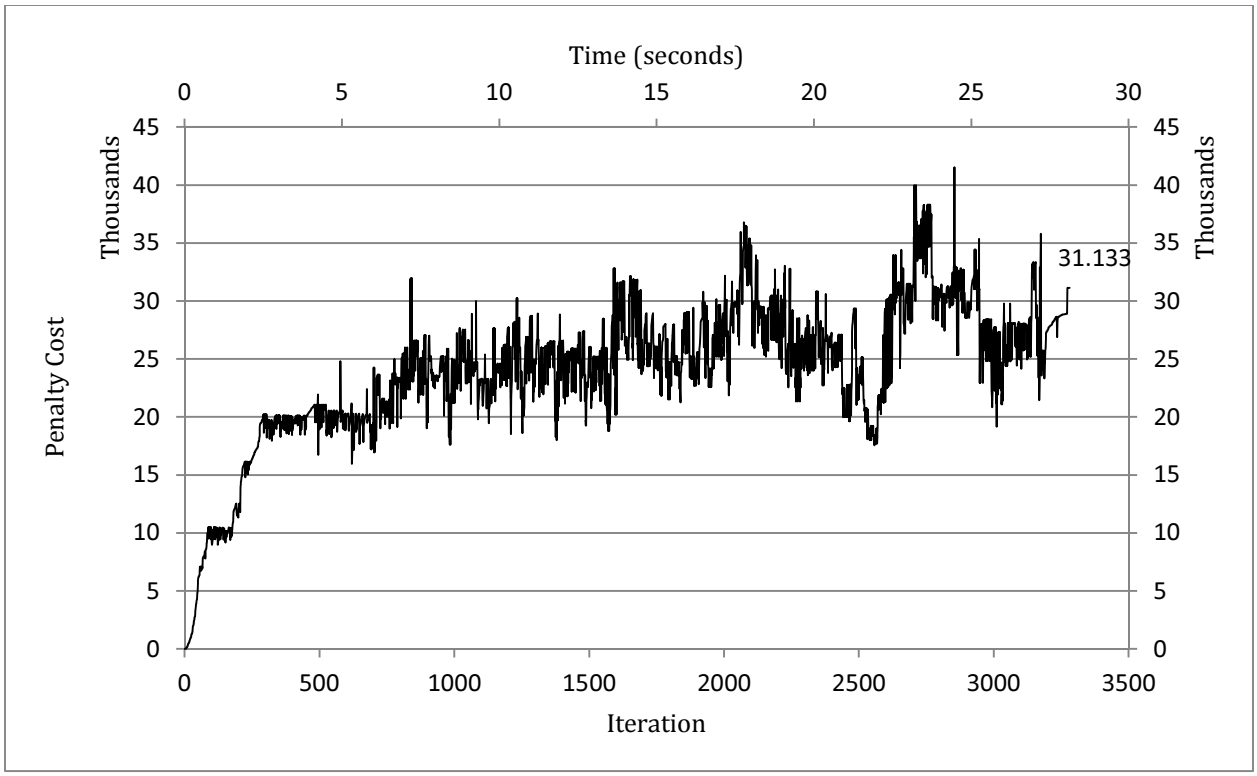


Figure 23 - Dataset 4 Construction Phase using Standard TS

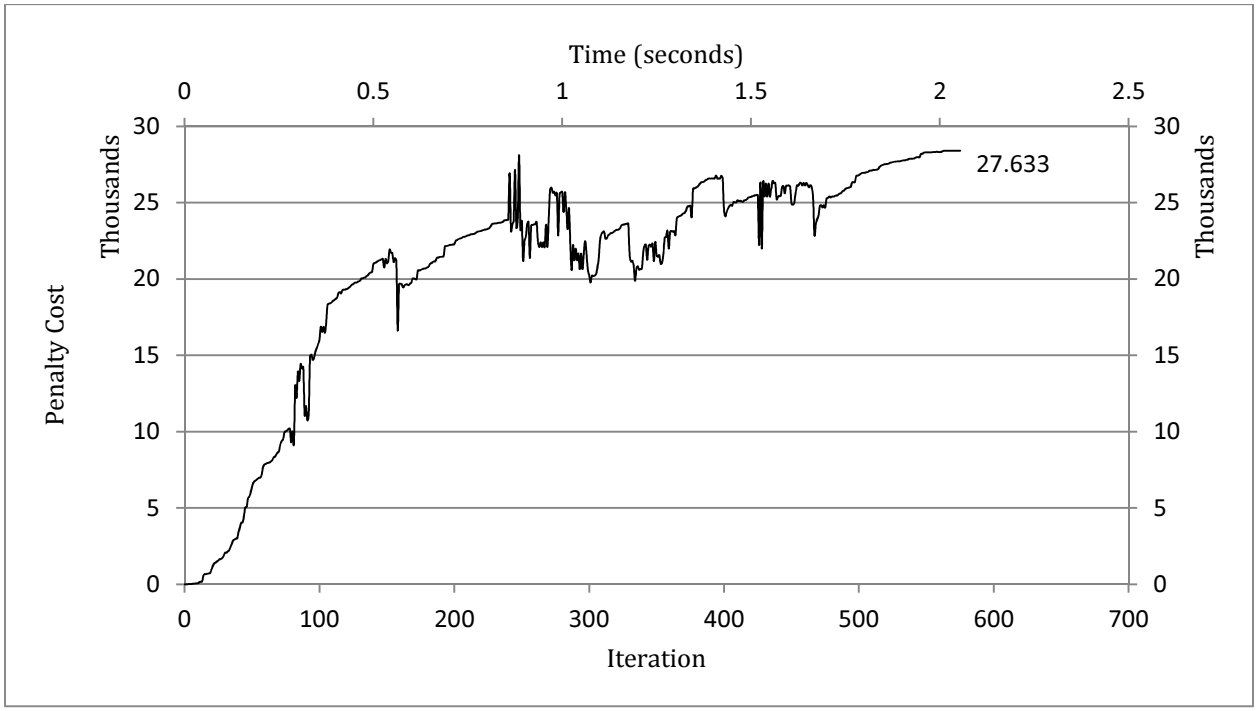


Figure 24 - Dataset 4 Construction Phase using TS with CD

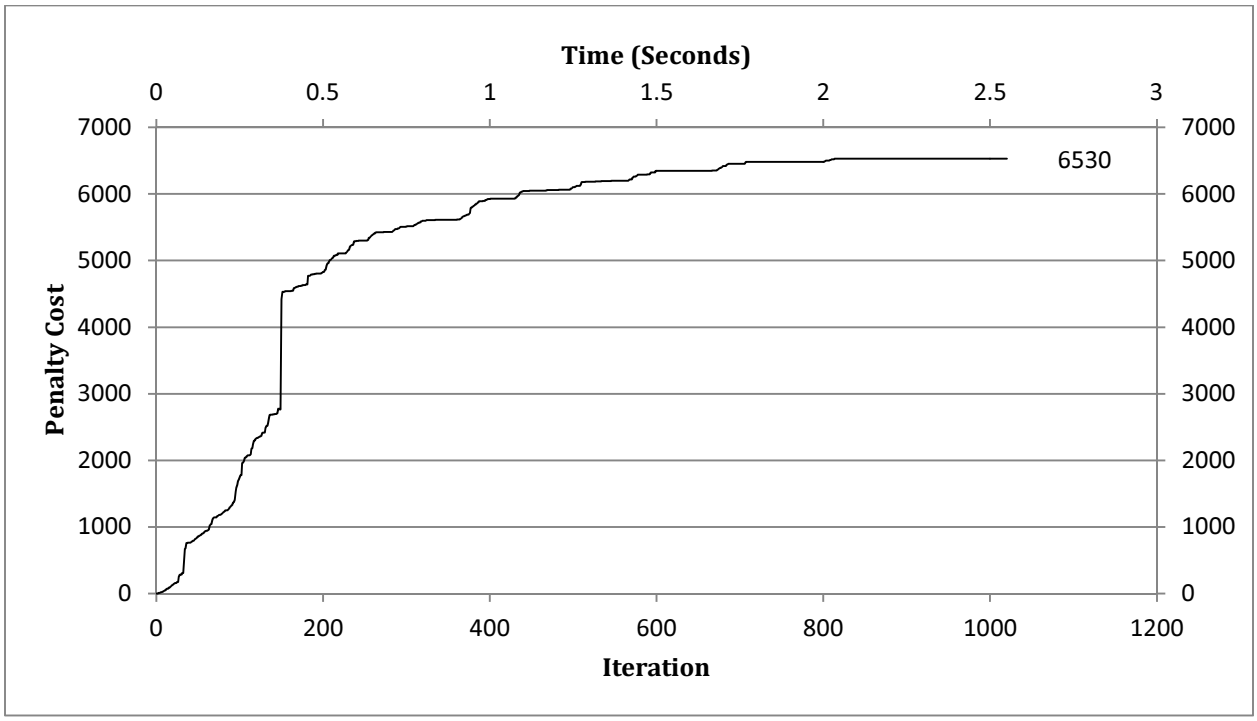


Figure 25 - Dataset 5 Construction Phase using Standard TS

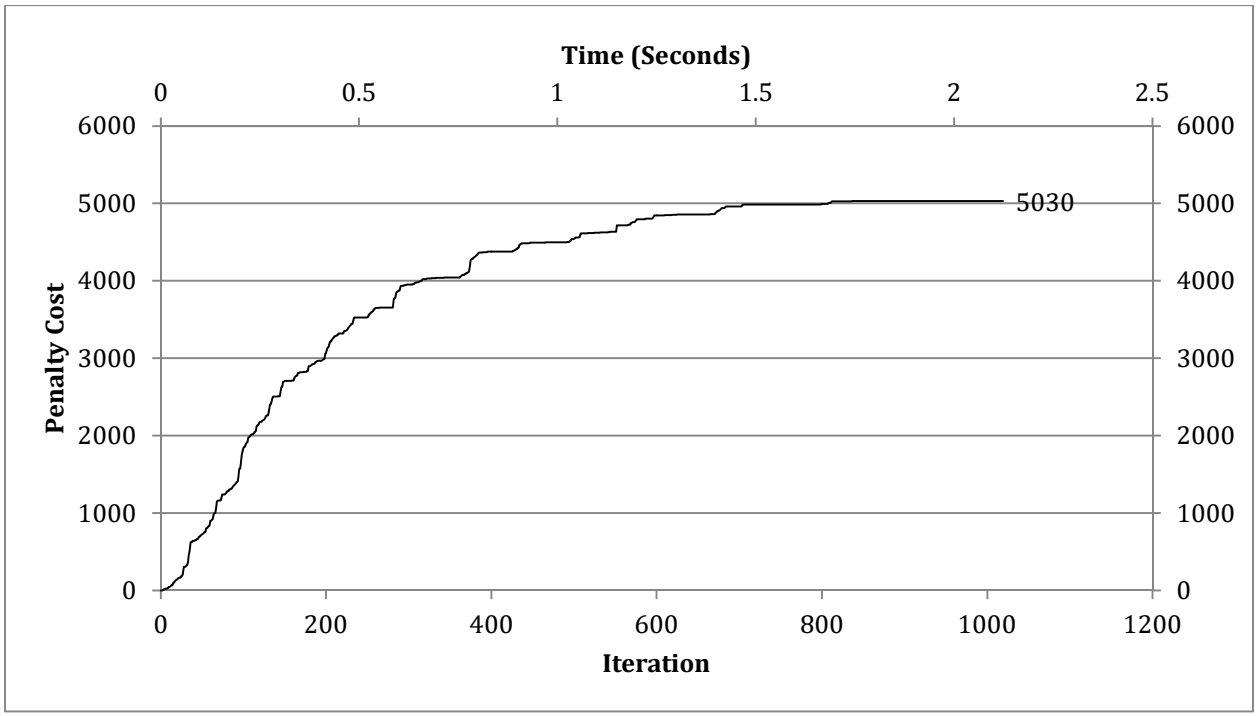


Figure 26 - Dataset 5 Construction Phase using TS with CD

7.2.4.2 Enhancement Phase Testing and Analysis

Now, we focus our attention on enhancement phase which launches immediately after construction phase reaches its first complete feasible solution. It tries to improve the quality of the solution found so far. Figures 27, 28 and 29 show the enhancement phase's best solution distribution history in sequential theme for the different algorithmic methods against time in dataset 8 and against iteration in datasets 1 and 6. From these figures, we clearly can notice that without preprocessing the first method tend to improve solutions' values within a relatively short time and it keeps improving almost very slowly.

Another visible notice is that method 1 seems to use less number of iterations which suggests that it employs these iterations cycles either in backtracking or accessing non efficient collections. Method 2 which also uses SA starts enhancing complete solution very early but then it ends with slightly outperforming method 1.

On the other hand, the last two methods, using GD flavours with preprocessing in place, spend some time to find the first improving solution after the first complete solution which also tends to be of, relatively, worse value than method 1 and 2. This is due to the nature of great deluge algorithm which only accepts an improving solution and a worse solution is also accepted if the quality of the solution is less than (for the case of a minimization problem) or equal to an upper boundary or "level" in

which during the search process, the "level" is iteratively updated by a constant decreasing rate.

It also means that, with the preprocessing phase in place, there will be more features, which means more effort to satisfy more constraints but also gaining better performance when looking up the different collections in particular area and also more careful exploration. For the inclusion of preprocessing phase, our proposed search algorithm's diversification of search to gather the whole search space proved important to find the global minimum quickly.

It is similarly noteworthy that MEGD performs slightly better than EGD in 8 out of 12 of the datasets. Figure 27 shows the results for dataset 1. It illustrates that methods 3 and 4 were close in terms of results in achieving best solution. This is also the case for method 1 and 2 although method 2 outperformed to some extent method 1. Method 3 reached a best solution value of 4185. The same pattern also appears in figure 28 and 29 where they show results for dataset 6 and dataset 8 where method 4 is marginally the winner in finding the best solution.

Generally, the algorithms might behave differently due to the different measurements enforced during the search process. However, the difference between SA, GD, EGD and MEGD algorithms lies in the acceptance criteria functionality that would make a difference on the limited solving time that was imposed on our benchmarking datasets. This is, nonetheless, might not be the case if we have relatively longer times

for several hours or days as all these algorithms are based on the stochastic local search and there will be always the possibility of achieving good results.

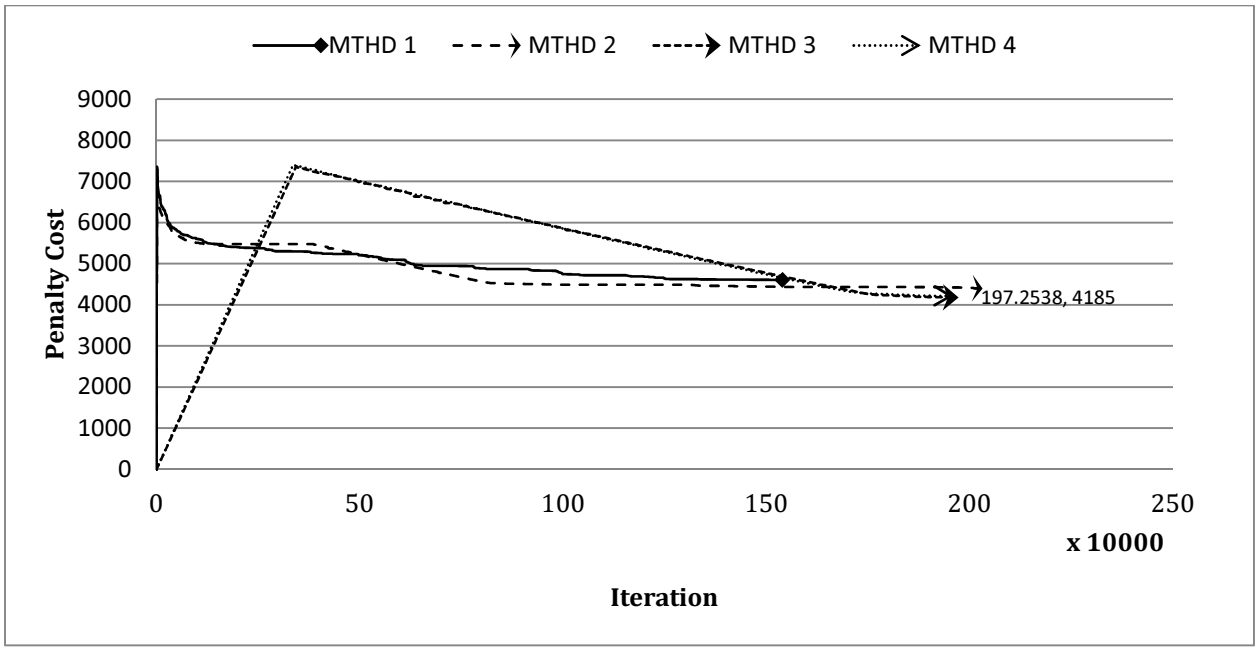


Figure 27 –Enhancement Phase Complete Solutions against Iterations - Dataset 1

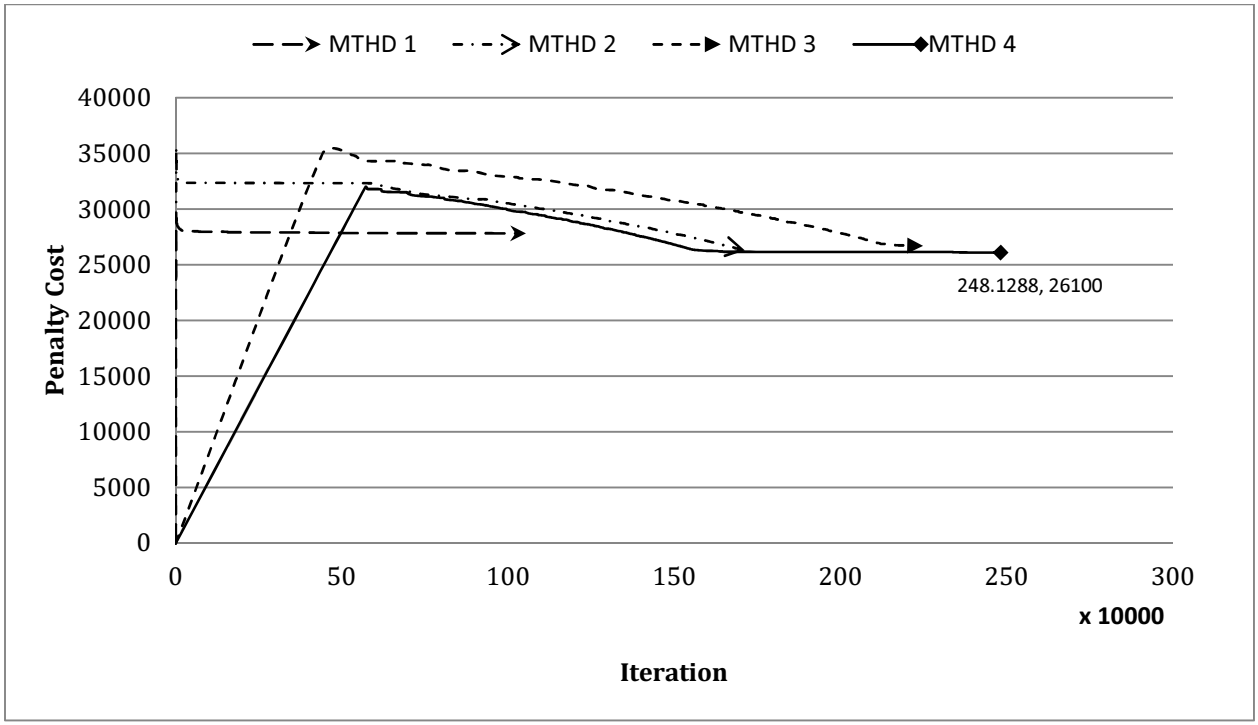


Figure 28 – Enhancement Phase Complete Solutions against Iterations - Dataset 6

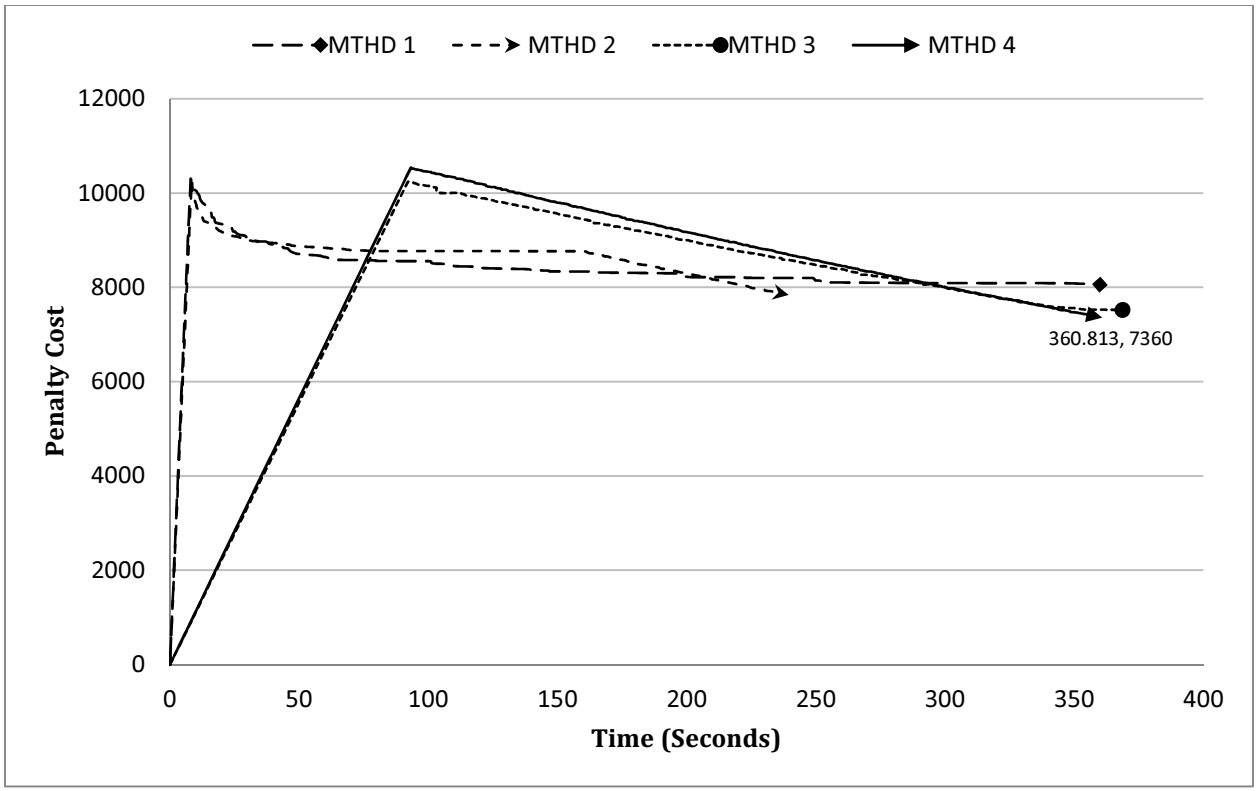


Figure 29 –Enhancement Phase Complete Solutions against Time - Dataset 8

7.3 Testing Results and Analysis

On the basis of results obtained by both construction phase and enhancement phase algorithms, we decided to experiment with two testing themes; sequential and parallel. Each of the datasets used in our testing phase has a previously discovered best known solution announced by ITC 2007. The five finalists of the examination track of the competition have applied different approaches. These approaches are:

- Muller [140] implemented a constraint-based solver, which constructs a complete solution, followed by a hill climbing and a great deluge approach for improving the solution.

- Gogos et al. [97] used a Greedy Randomized Adaptive Search Procedure, followed by simulated annealing, and integer programming.
- Atsuta et al. [10] developed a constraint satisfaction solver combined with Tabu search and iterated local search.
- De Smet [71] has his own solver, namely Drools Solver, which is a combination of Tabu search and the JBoss Drools rules engine used for the evaluation.
- Pillay [146] used a heuristic method that is inspired by cell biology.

In order to assess the relative effects of the four algorithms variations, we performed experiments using the same settings that we considered for all algorithms involved, which we previously detailed in table 6. During experiment runs, we managed to achieve an outstanding 98% success in reaching complete feasible solution on all instances in all attempts in both sequential and parallel themes. The remaining 2% were only in dataset 4 and 12.

For each method trials we performed 11 individual runs on each of the 12 competition instances, using the time limit specified by the competition benchmarking program as our stopping criteria, which equated to 362 seconds and 377 seconds of CPU time on the two testing machines. The same timeout value on each machine is used for all of the 12 datasets. In all cases, we logged out all best solution values history along with times and iterations where these best solution values are discovered. The settings of the algorithms have remained the same

throughout the experiment for the purpose of going in line with ITC 2007 rules. One of our objectives in testing phase is to represent different algorithm variations that are composed of different algorithms and compare them to the performance of ITC 2007 results. The expectation was also set for the results to be reasonably comparable if not better than ITC 2007 exam track results.

In table 9, results for sequential theme show best solution value (lowest penalty cost) and average of best solution values for each variant. The main observation made was that when searching without preprocessing, performance degrades relatively to when using preprocessing phase. Only the first method did not use preprocessing and if we look first at the performance of its algorithms in comparison to ITC 2007 results we will notice that it comes in the second place in 10 out of 12. This is the case for all datasets except datasets 10 and 12. TS with CD + HC+ SA with no preprocessing is the worst algorithm variant in our testing and it comes in the second place in most datasets in comparison to ITC 2007 results.

The other three algorithms variants performed better. Only when we used GD algorithm extensions (EGD and MEGD), we started to see results that overtake ITC 2007 results. Our approach gets 8 out of 12 datasets as best results. These results are split between EGD and MEGD evenly with 4 best results each.

In order to obtain a fair comparison, it is worth noticing that the performance loss is on average about 7% between SA with preprocessing and MEGD with preprocessing,

whereas it is about 10% if the preprocessing phase is not implemented with SA. Moreover, one can also notice that the gap between the two methods becomes smaller with higher conflicts density problems, and that the behavior of the methods with pre-processing phase implemented is more stable with respect to SA with no preprocessing phase.

All in all, EGD and MEGD performed much better than SA with preprocessing phase not to mention SA with no preprocessing. Another reasonable comparison is that all of our methods performed well in comparison to ITC 2007 results in best solution values and in best average values.

Table 10 shows results for parallel theme. Two machines were involved in the testing. Because of ITC 2007 rules, we do not compare them to its results but rather to our sequential theme results. The results show that there is a significant net positive effect on the overall solutions quality when the algorithms are run in parallel. Parallel runs outperformed sequential runs in terms of best solutions values and best average solution values in 8 out of 12 datasets.

Instance #	1	2	3	4	5	6	7	8	9	10	11	12
T. **Muller**	4370	**400**	10049	18141	2988	26585	4213	7742	**1030**	16682	34129	5535
C. **Gogos**	5905	1008	13771	18674	4139	27640	6572	10521	1159	-	43888	-
M. **Atsuta**	8006	3470	17669	22559	4638	29155	10473	14317	1737	15085	-	**5264**
G. **Smet**	6670	623	-	-	3847	27815	5420	-	1288	**14778**	-	-
N. **Pillay**	12035	2886	15917	23582	6860	32250	17666	15592	2055	17724	40535	6310
MTH1	TS With CD + HC + SA + No Preprocessing											
Best	4608	558	10491	22023	3407	27825	4697	8060	1087	15640	34171	6216
Avg	4805.8	574.8	11135.1	24210.6	3610.9	29685.0	4860.7	8302.4	1181.5	18638.3	37093.2	6944.8
MTH2	TS With CD + SA + Preprocessing											
Best	4518	455	10415	18175	3229	26305	4408	7843	1055	15504	33229	5381
Avg	4798.8	495.6	11376.9	23184.4	3699.6	28779.0	4742.3	8321.6	1148.5	18163.1	36928.4	6667.2
MTH3	TS With CD + EGD + Preprocessing											
Best	**4185**	415	10002	**17662**	**2693**	26705	**4149**	7524	1041	15745	32333	5857
Avg	4404.1	432.3	10363.2	20457.1	2967.2	27605.5	4309.0	8143.0	1086.2	18430.4	38412.9	6582.3
MTH4	TS With CD + MEGD + Preprocessing											
Best	4218	420	**9335**	18658	2718	**26100**	4181	**7360**	1050	14918	**31177**	5544
Avg	4395.0	433.5	10118.6	21722.9	2836.4	27166.8	4294.0	7632.7	1109.6	17022.2	33608.6	6364.4

Legend: **1234** *Best Value*; 1234 *Best Average*

Table 9 - Comparison of Performance of Sequential Experiment Results With ITC 2007 Results

Instance #	1	2	3	4	5	6	7	8	9	10	11	12
MTH1	TS With CD + HC + SA + No Preprocessing											
SQ	4608	558	10491	22023	3407	27825	4697	8060	1087	15640	34171	6216
PR	4479	455	9724	18624	3323	26485	4366	7831	1041	15269	32374	5580
SQ AVG	4805.8	574.8	11135.1	24210.6	3610.9	29685.0	4860.7	8302.4	1181.5	18638.3	37093.2	6944.8
PR AVG	4759.3	485.5	11239.1	23008.0	3702.6	28587.7	4697.5	8178.0	1125.1	18286.7	37573.5	6502.9
MTH2	TS With CD + SA + Preprocessing											
SQ	4518	455	10415	18175	3229	26305	4408	7843	1055	15504	33229	5381
PR	4448	421	9841	20205	3121	26280	4316	7745	1047	15285	31602	5461
SQ AVG	4798.8	495.6	11376.9	23184.4	3699.6	28779.0	4742.3	8321.6	1148.5	18163.1	36928.4	6667.2
PR AVG	4466.8	437.2	10947.0	22547.5	3514.5	27086.5	4568.6	7945.4	1118.7	18313.3	35097.0	6278.3
MTH3	TS With CD + EGD + Preprocessing											
SQ	**4185**	415	10002	**17662**	2693	26705	4149	7524	1041	15745	32333	5857
PR	4374	416	10066	18379	2799	26125	4091	7461	1039	14941	29830	**5326**
SQ AVG	4404.1	432.3	10363.2	20457.1	2967.2	27605.5	4309.0	8143.0	1086.2	18430.4	38412.9	6582.3
PR AVG	4382.6	432.7	10800.6	20759.6	2964.3	27113.0	4255.4	7598.2	1099.8	17381.6	33463.1	6289.0
MTH4	TS With CD + MEGD + Preprocessing											
SQ	4218	420	**9335**	18658	2718	26100	4181	**7360**	1050	14918	31177	5544
PR	4264	**410**	9473	18026	**2658**	**25935**	**4066**	7401	**1029**	**14850**	**29507**	5434
SQ AVG	4395.0	433.5	10118.6	21722.9	2836.4	27166.8	4294.0	7632.7	1109.6	17022.2	33608.6	6364.4
PR AVG	4402.8	432.7	10250.0	20547.5	2926.0	26744.2	4240.9	7752.5	1072.2	16567.1	33357.8	6036.0

Legend **1234** *Best Value*
1234 *Best Average*

Table 10 - Comparison of Performance between Sequential and Parallel Experiment Results

These results indicate that regarding used algorithms in both sequential and parallel themes, TS with CD and MEGD is the better algorithm in terms of achieving best solution values and average solution values, returning 9 out of 12 in best solution values and 8 out of 12 in best average solution values. However, for the two algorithms that Simulated Annealing and Hill Climbing are involved in, parallel approach achieved better results in all datasets and MEGD only slightly outperformed EGD.

Figure 30 and 31 show the full distribution history for datasets 11, 12 respectively. Graph in figure 30 uses MEGD algorithm and graph in figure 31 uses and EGD algorithm variant in sequential and parallel themes. The graph lustrates how the parallel and sequential approach reached its best value. Nevertheless, this is not to indicate that the sequential approach performed worse but rather because of the cooperating behaviour of the parallel approach, it was able to outperform most of the sequential results.

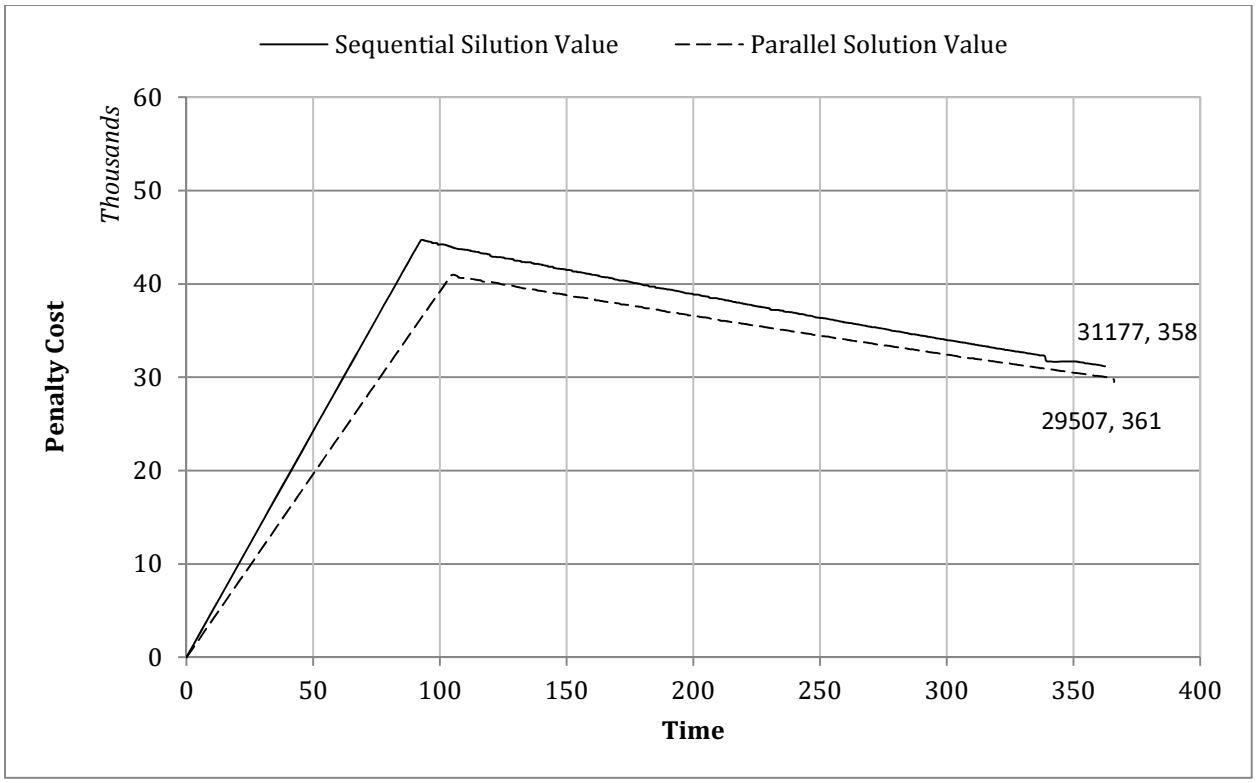


Figure 30 - Solution Value History for dataset 11 in MEGD (sequential and Parallel)

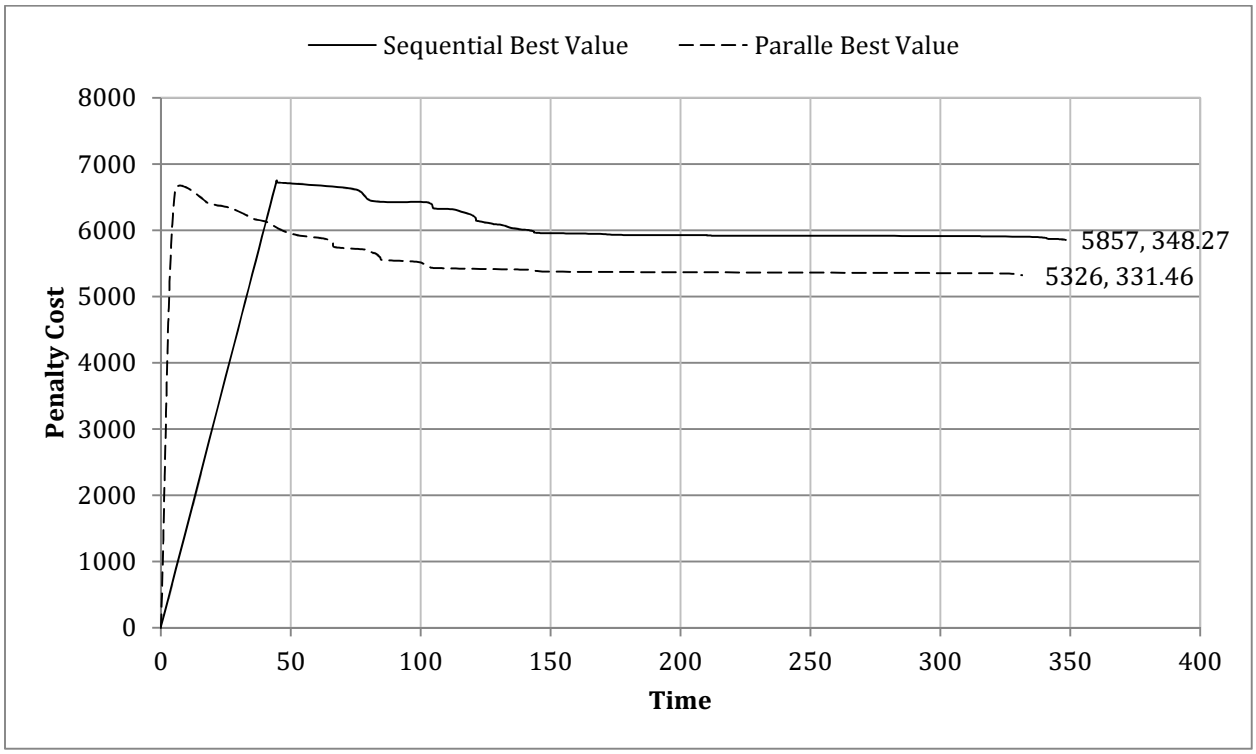


Figure 31 - Solution Value History for dataset 12 in EGD (sequential and Parallel)

Table 11 shows parallel theme's machines contribution to best solution values history in all algorithm variants which relatively indicates a very good load balance that we were assuming in using a solver in parallel co-operating environment.

	DS1	DS2	DS3	DS4	DS5	DS6	DS7	DS8	DS9	DS10	DS11	DS12
M1	39.3%	72.5%	43.6%	62.0%	34.7%	76.2%	66.5%	58.3%	71.0%	26.1%	38.9%	54.2%
M2	60.7%	27.5%	56.4%	38.0%	65.3%	23.8%	33.5%	41.7%	29.0%	73.9%	61.1%	45.8%

Table 11 - Parallel theme's machines contribution to best solution values history in all methods

The results from experiments, on benchmark datasets, show that our approach is superior over other methods based on the classical local search methods. On average it produces very good and competitive results compared to other methods used by ITC 2007 winners.

7.4 Conclusion

In this chapter we went through experiments and testing phase where we examined our proposed approach which is an implementation of hybrid multiphase metaheuristics search algorithm based on local search algorithms. Our case study was taken from International Timetabling Competition 2007 (Exam Track). The same algorithm is utilized as a parallel algorithm, by coordinating multiple instances of a component solver using MPI protocol. After equipping a constraint solver with the best solution exchange mechanism, the proposed approach was sufficient to coordinate two of these solvers instances to perform parallel search. Experiments

showed that better results were obtained using only 2 machines in comparison to the same algorithm variant running in sequential mode, which indicates a good load balance.

Finally, we conclude that:

- The IDS Solver's ability to be configurable and its pluggable features made it very easy to change, extend and plug in new algorithms.
- Once a solver is able to reach good results, the fact of the matter states that building parallel constraint-based solver becomes a concept of a component-based software engineering and architecture.
- The pre-processing phase that we proposed was able to improve results marginally and this could be used in problems where stopping criteria is limited by short times.
- MEGD is an extension to GD which only shows its best when we need to skip local optimum sooner.
- Separating search computation from instances coordination through a known protocol, such as MPI, instead of implementing fully parallel algorithm is a feasible approach.
- Satisfying Results were obtained both in sequential and parallel themes which fulfils our initial goal of competing with ITC 2007 results.

8. Conclusions and Future Directions

There were five objectives of the research presented in this thesis. Firstly is, to reconstruct, rebuild and validate a C# based COP Solver that uses backtracking with look-ahead forward checking algorithm to be extended to use preferences to solve teaching assignment problem.

Secondly is, to define, describe and construct a local search based COP solver as a generic solver for any constraints satisfaction problem to be modelled and extended to use multi-phase parallel local search algorithms with hybrid metaheuristics to solve examination timetabling problem.

Thirdly is, to verify experimentally both frameworks and by using a well-known benchmarking datasets with the second solver to produce reasonable and competitive results and using datasets from the computer science department with the first solver.

Fourthly is, to explore the applicability of Microsoft technologies to real world scientific domains with a focus on timetabling applications.

Finally is, to extend existing algorithms and methodologies to create and build a generic solver that can be pluggable and extensible for modelling and solving specific problems.

We believe that we successfully achieved most of these objectives and to some extent partially successfully others. This can be seen in the number of contributions we made with this thesis. However, more future research can make use of the current research as basis for more investigation and improvements.

8.1 Research Contribution

The accomplished research to attain the above objectives can be summarised in the following research contributions detailed in the following points.

1. An extension approach to the backtracking with look-ahead forward checking method that adopts weighted partial satisfaction of soft constraints that has been implemented to the development of an automated teaching assignment timetabling system. Results showed that feasible schedules were obtained for real data sets, including professors' preferences without the need for a huge computational effort. The original solver was meant to solve integer based variables problems but for problems with hard constraints. We have extended the solver to adopt soft constraints. We think that this approach can be implemented in similar problems like Exam Supervision scheduling. Based on the gathered results, we concluded that this approach is computationally feasible.
2. A configurable, pluggable and easy to use local search solver. The solver is named *"Incremental Dynamic Search Solver"*, which is a generic local search solver that starts with feasible solution and then improves the quality of the constructed solution. IDS solver is used to model and solve the exam timetabling problem.

The fundamental idea behind the development of IDS Solver is to consolidate Local Search techniques and features in one place. The framework provides a model for the design and implementation of Local Search algorithms and reveals numerous advantages concerning implementing the algorithm from start to finish in terms of code reuse, organisation and methodology. The idea is to abstract away the essential infrastructure core and details of local search algorithms and to leave only the problem-specific development details to the user. IDS Solver makes use of the best implementation of design patterns and software best practices to gain a better computational efficiency. The system allows the user to optionally pick and mix any local search techniques and to generate and experiment new combinations of features. We introduced the main layers and architecture of IDS Solver that composed the basis for the design, model and solving of large examination timetabling.

3. A parallel technique as a plug-in functionality using *"Message Passing Interface, MPI"* method which has been used in the proposed IDS solver to solve COP problems using more than one solver instance that can be hosted on the same machine or on different machines. The technique is responsible for communicating best solutions among IDS solver instances collaboration which perform the core of the optional parallel problem solving. The approach was used in the parallel theme in our testing to solve Large Exam Timetabling Problem (ITC 2007 Exam track benchmarking datasets).

4. A multi-phase parallel hybrid metaheuristic approach to solve exam timetabling problem. The proposed approach was used to solve exam timetabling problem using hybrid metaheuristics that is optionally can be run in parallel using the proposed MPI functionality utilizing different instances hosted on one or different machines. It employs different metaheuristics search methods; a modified version of tabu search, hill climbing, simulated annealing, and different flavours of great deluge.

   The selected heuristics process in sequence to produce a good solution for the current state of the problem. The whole hybrid heuristics approach is configurable and able to manage and control its heuristics without having a domain pre-knowledge of the exam timetabling problem.

5. A pre-processing stage in solving examination timetabling problem which involves two phases prior to commencing local search; problem collections ordering and unspecified constraints discovery technique that is used to discover all hard constraints that are not usually explicitly defined in the problem. The technique uses a full representational graph of the problem. The goal of the pre-processing is to enhance the overall search process.

6. A novel approach that modifies Tabu Search algorithm and uses a dictionary data structure mechanism to represent conflicts. The tabu list does not contain operators or moves that are problem specific. It only needs to store the conflicted moves along with accumulated number of conflicts it caused. In this approach,

instead of recording recent moves, we record moves with an accumulated count of conflicts that caused. The idea behind conflict dictionary is to work as memory storage for previous clashes to avoid their potential reappearance in future. The approach is introduced as a construction phase to achieve a partial or complete initial feasible solution.

7. A modification to the Extended Great Deluge (EGD) method to reduce the amount of time wasted on local optimum. The proposed modified method is named MEGD and used to solve large examination problem. The approach is used during search to eliminate some of the time wasted in local optimum based on certain conditions.

8.2 Future Research

Although our research covered many aspects of the timetabling problem in general, we still have some ideas and concerns that we need to touch on and search in depth. The following are some of these ideas.

1. IDS Solver is implemented using Microsoft .NET framework and developed using C# language under Microsoft Windows operating system. The solver could be easily ported to the Linux version of .NET framework which called “Mono”. In our case, the testing phase took place on two machines with single-CPU (multi-core) which did not test the parallel approach to its full extent. We think that a “mono” version would open the door for more testing on some the powerful machines as most universities and research centres have possession of Linux based machines

with 16-32-64 CPUs that could leverage several solver instances running and cooperating on the same machine which might lead to better results.

2. The solver uses several parameter values that can be injected using custom configuration settings. These values that we unified and used for the entire testing life cycle performed very well and we are satisfied with the results we achieved. However, for the sake of objectivity, we should indicate that we have not performed a full parameter fine tuning runs on all of the algorithms used in our experiment, and those parameters values have been set to be appropriate values according to the literature and we only preliminary examined them in a pre-testing runs. We believe that if algorithms use the right settings for its parameters, it might lead to even better results. This is a whole research topic that might be investigated more and we recommend that this might be done in a future work.
3. The parallel approach may be improved in Great Deluge and its variants algorithms by letting one instance work from lower bound to the current solution and let the other work from current solution to the upper bound for another instance. If we have more instances we can divide the range from lower bound to upper bound accordingly. Once a better solution is found we can propagate solution to other instances and restart again using the same schema.
4. IDS Solver is extendable and pluggable approach and we would like to see other local search methods applied and used within the same framework and tested on

other problems other than timetabling problem in an attempt to prove its potential of being a real generic solver.

5. There are certain situations where local solvers have trouble dealing with. For instance, there are a variety of situations that cause IDS solver to sometimes get stuck in regions of the variable space where they constantly make small steps and slight progress. The idea of switching to another part of the search space and escape local optimum is proved to be feasible as this is already done in our approach of MEGD algorithm and it would be beneficial to see the impact of the same approach on other local search methods throughout the optimization process.

Bibliography

[1] Aarts, E. and Lenstra, J. K., "Local search in combinatorial optimization," 1997.

[2] Aarts, E.H.L., Korst, J.H.M, *Simulated Annealing and Boltzmann Machines*. Chichester: Wiley, 1989.

[3] Aarts, Emile and Lenstra, Jan Karel, *Local Search in Combinatorial Optimization*.: Wiley, 1997, vol. Wiley Series in Discrete Mathematics & Optimization.

[4] Abdennadher S. and Marte M., "University Timetabling Using Constraint Handling Rules," *Journal of Applied Artificial Intelligence*, no. Special Issue on Constraint Handling Rules, 1999.

[5] Abramson, D. A., Dang, H., and Krisnamoorthy, M., "Simulated annealing cooling schedules for the school timetabling problem," *Asia Pacific Journal of Operational Research*, vol. 16, pp. 1-22, 1999.

[6] Abramson, D. A., Dang, H., and Krisnamoorthy, M., "Simulated annealing cooling schedules for the school timetabling problem," *Asia-Pacific Journal of Operational Research*, vol. 16, pp. 1-22, 1999.

[7] Appleby, J. S., Blake, D. V. and Newman, E. A., "Techniques for producing school timetables on a computer and their application to other scheduling problems," *The Computer Journal*, vol. 3, pp. 237-245, 1960.

[8] Applegate D., Cook W., "A computational study of the job-shop scheduling problem," *ORSA Computing Journal*, vol. 3, no. 2, p. 149.156, 1991.

[9] Asmuni, H., Burke, E. K., Garibaldi, J. M., and McCollum, B., "Fuzzy multiple heuristic orderings for examination timetabling," in *Lecture notes in computer science, Proceedings of the 5th international conference on the practice and theory of automated timetabling, PATAT 2004,* , vol. 3616, Pittsburg, PA, USA, 2005, pp. 334-353.

[10] Atsuta, T., Nonobe, M., and Ibaraki, , "Itc2007 track 1: An approach using a general csp solver (Technical report)," 2008.

[11] Awad, R. and Chinneck, J., "A multi-stage evolutionary algorithm for the timetable problem," *The IEEE Transactions on Evolutionary Computation*, vol. 3, no. 1, pp. 63-74, 1999.

[12] Awad, R. and Chinneck, J., "Proctor Assignment at Carleton University," *Interfaces*, vol. 28, no. 2, pp. 58-71, 1998.

[13] Balakrishnan, N., Lucena, A., and Wong, R. T., "Scheduling examinations to reduce second order conflicts," *Computers and Operations Research*, vol. 19, pp. 353-361, 1992.

[14] Bartak, R., "Dynamic Constraint Models for Planning and Scheduling Problems," *New Trends in Constraints*, vol. LNAI 1865, pp. 237-255, 2000.

[15] Bartak, R., "Dynamic Constraint Models for Planning and Scheduling Problems," *New Trends in Constraints*, vol. LNAI 1865, pp. 237-255, 2000.

[16] Bayardo R. Jr., Schrag R., "Using CSP look-back techniques to solve exceptionally hard SAT instances," in *CP-96*, 1996, pp. 46–60.

[17] Brailsford, S. C., Potts, C. N., and Smith, B. M., "Constraint satisfaction problems: algorithms and applications," *European Journal of Operational Research*, vol. 119, pp. 557-581, 1999.

[18] Broder, S., "Final examination scheduling," *Communications of the ACM*, vol. 7, pp. 494–498, 1964.

[19] Burke E. K, MacCathy, B. Petrovic, S. and Qu, "Knowledge discovery in a hyperheuristic for course timetabling using case-based reasoning," *PATAT'02*, vol. 14, pp. 90-103, 2002.

[20] Burke E. K. , Kendal G., McCollum B. and McMullan P., "Constructive versus Improvement Heuristics: An Investigation of Examination Timetabling," in *3rd Multidisciplinary International Scheduling Conference: Theory and Applications*, Paris, France, 2007.

[21] Burke E. K., Petrovic, S. and Qu. R., "Case-based heuristic selection for examination timetabling," in *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution And Learning*, 2002, pp. 277-281.

[22] Burke E., Kingston J., Jackson K. and Weare R., "Automated University Timetabling: The State of the Art," *The Computer Journal*, vol. 40, no. 9, pp. 565-571, 1997.

[23] Burke, E. K. and Landa Silva, J. D., "The design of memetic algorithms for scheduling and timetabling problems," *Recent studies in Memetic Algorithms, Studies in Fuzziness and Soft Computing*, vol. 166, pp. 289-312, 2004.

[24] Burke, E. K. and Newall, J. P., "Enhancing timetable solutions with local search methods," in *Lecture notes in computer science: Vol. 2740. Proceedings of the 4th international conference on the practice and theory of automated timetabling, PATAT 2002*, Gent, Belgium, 2003, pp. 195-206.

[25] Burke, E. K. and Newall, J. P., "Solving examination timetabling problems through adaption of heuristic orderings," *Annals of Operations Research*, vol. 129, no. 1-4, pp. 107-134, 2004.

[26] Burke, E. K. and Petrovic, S., "Recent research directions in automated timetabling," *European Journal of Operational Research*, vol. 140, pp. 266–280, 2002.

[27] Burke, E. K. and Ross P., "Some combinatorial models for course scheduling.," *PATAT'95*, vol. LNCS 1408, pp. 296-308, 1995.

[28] Burke, E. K., Bykov, Y., Newall, J. and Petrovic, S., "A Time-predefined local search approach to exam timetabling problems," *IIE Transactions*, vol. 36, no. 6, pp. 509-528, 2004.

[29] Burke, E. K., Eckersley, A. J., McCollum, B., Petrovic, S. and Qu, R., "Analysing similarity in exam timetabling," in *Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling*, 2004, pp. 89-106.

[30] Burke, E. K., Elliman, D. G., Ford, P. H. and Weare, R. F., "Examination timetabling in British universities - a survey. In: Practice and Theory of Automated Timetabling," *Lecture Notes in Computer Science*, vol. 1153, pp. 76-92, 1996.

[31] Burke, E. K., Hart, E., Kendall, G., Newall, J., Ross, P. and Schulenburg, S.,. Dordrecht: Kluwer, 2003, ch. 16, pp. 457–474.

[32] Burke, E. K., Hart, E., Kendall, G., Newall, J., Ross, P. and Schulenburg, S., "Hyper-heuristics: An emerging direction in modern search technology," in *Handbook of Metaheuristics*. Kluwer, Durdrecht, 2003, ch. 16, pp. 457-475.

[33] Burke, E. K., Kingston, J., and de Werra, D., "Application to Timetabling," in *Handbook of graph Theory*. London: Chapman Hall, 2004, pp. 445-474.

[34] Burke, E. K., Mac Carthy, B., Petrovic, S. and Qu, R., "Multiple retrieval case based reasoning for course timetabling problems," *Journal of Operational Research Society*, vol. 57, no. 2, pp. 148-162, 2005.

[35] Burke, E. K., MacCarthy, B., Petrovic S. and Qu., R., "Case based reasoning in course timetabling: An attribute graph approach," in *Proceedings of the 4th. International Conference on Case-Based Reasoning*, vol. 2080, 2001, pp. 90-104.

[36] Burke, E. K., Newall, J. P. and Weare, R. F., "A memetic algorithm for university exam timetabling. In: Practice and Theory of Automated Timetabling, E. K. Burke and P. Ross, eds.," *Lecture Notes in Computer Science Volume 1153, Springer-Verlag*, pp. 241-250, 1996.

[37] Burke, E. K., Newall, J. P. and Weare, R. F., "Initialisation strategies and diversity in evolutionary timetabling," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 1, pp. 81-103, 1998.

[38] Burke, E.K. and Ross, P. (eds), "Practice and Theory of Automated Timetabling," in *1st International Conference on the Practice and Theory of Automated Timetabling, Lecture Notes in Computer Science*, Berlin Heidelberg, 1996.

[39] Burke, E.K., and Erben, W., "Practice and Theory of Automated Timetabling III, Third International Conference, PATAT 2000," in *Selected papers, Lecture Notes in Computer Science*, 2001.

[40] Burke, E.K., and M.W. Carter (eds.), "Practice and Theory of Automated Timetabling II, Second International ConferenceConference, PATAT '97," in *Selected papers, Lecture Notes in Computer Science*, 1998.

[41] Burke, E.K., and M.W. Carter (eds.), "Proceedings of the Second International Conference on the Practice and Theoory of Automated Timetabling," , University of Toronto, Canada., 1997.

[42] Burke, E.K., and W. Erben (eds.), "Practice and Theory of Automated Timetabling III, Third International Conference, PATAT 2000," in *Selected papers, Lecture Notes in Computer Science* , 2001.

[43] Burke, E.K., and W. Erben (eds.), "Proceedings of the Third International Conference on the Practice and Theory of Automated Timetabling," , Fachhochschule Konstanz, Germany, 2000.

[44] Burke, E.K., MacCathy, B., Petrovic, S. and Qu, R., "Structured case in case-based reasoning-re-using and adapting cases for timetabling problems," *Knowledge-Based Systems*, vol. 13, pp. 159-165, 2000.

[45] Burke, E.K., McCollum, B., Meisels, A., Petrovic, S. and Qu, R., "A graph-based hyperheuristic for educational timetabling problem," *European Journal of Operational Research*, pp. 1-16, 2006.

[46] Cambazard, H., Demazeau, F., Jussien, N. and David P., "Interactively solving school timetabling problems using extensions of constraint programming," in *Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling*, 2004, pp. 107-124.

[47] Cambazard, Hadrein et al, "Interactively Solving School Timetabling Problems Using Extensions of Constraint Programming," *PATAT* , pp. 190-207, 2004.

[48] *Canadian Oxford Dictionary*, 2nd ed.: Oxford University Press, 2004.

[49] Caprara, A., Fischetti, M., Guida, P. L., Monaci, M., Sacco, G. and Toth, P., "Solution of real-world train timetabling problems," in *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*, 2001, pp. 1057-1067.

[50] Caramia, M., Dell'Olmo, P., and Italiano, G. F., "New Algorithms For Examination Timetabling," *Lecture notes in computer science, Algorithm Engineering 4th International Workshop, WAE 2000, Saarbrücken, Germany*, vol. 1982, pp. 230-241, 2001.

[51] Carter, M.W., "A Survey of Practical Applications of Examination Timetabling Algorithms," *Operations Research Society of America*, vol. 34, no. 2, pp. 193-202, 1986.

[52] Carter, M. W. and Laporte, G., "Recent developments in practical examination timetabling. In: The Practice and Theory of Automated Timetabling," *Lecture Notes in Computer, VOl. 1153, Springer-Verlag*, pp. 3-19, 1996.

[53] Carter, M. W., Johnson, D. G. , "Extended partition initialization in examination timetabling," *The Journal of the Operational Research Society*, vol. 52, pp. 538-544, 2001.

[54] Carter, M. W., Laporte, G. and Lee, S. Y., "Examination timetabling: algorithmic strategies and applications," *Journal of the Operational Research Society*, vol. 47, no. 3, pp. 373-383, 1996.

[55] Carter, M.W. and Laporte, G., "Recent developments in practical course timetabling," *PATAT'97*, vol. LNCS 1408, pp. 3-19, 1997.

[56] Casey, S. and Thompson, J., "GRASPing the examination scheduling problem," *Practice and Theory of Automated Timetabling*, vol. 2740, pp. 232 –244, 2003.

[57] Chan, C. K., Gooi, H. B., & Lim, M. H., "Co-evolutionary algorithm approach to a university timetable system," in *The 2002 congress on evolutionary computation*, vol. 2, Honolulu, HI, USA, 2002, pp. 1946–1951.

[58] Choco Solver (2010, Oct.). [Online]. http://www.emn.fr/z-info/choco-solver/

[59] Cole, A. J., "The Preparation of Examination Timetables Using a Small Store Computer," *Computer Journal*, vol. 7, pp. 117-121, 1964.

[60] Colorni, A., Dorigo, M. and Maniezzo, V., "Metaheuristic for highschool timetabling," *Computational Optimisation and Applications*, vol. 9, pp. 275-298, 1998.

[61] Comm, C. and Mathaisel, D., "College course scheduling. A market for computer software support," *Journal of Research of Computing in Education*, vol. 21, pp. 187-195, 1988.

[62] Computer Sceince Department, Queens University, Belfast, Northern Ireland. (2007) International Timetabling Competition 2007. [Online]. http://www.cs.qub.ac.uk/itc2007/

[63] Cooper T. B. and Kingston J. H., "The Complexity of Timetable Constrcution Problems," *The Practice and Theory of Automated Timetabling*, pp. 283-295, 1996.

[64] Corne, D., Dorigo, M. and Glover, F., "New Ideas in Optimzation," 1999.

[65] Corne, D., Ross, P. and Fang, H., "Evolving Timetabling," *The Paractical Handbook of Genetic Algorithms*, vol. 1, pp. 219-276, 1995.

[66] Csima, J. and Gotlieb, C. C., "Tests on a computer method for construction school timetables," *Communication of the ACM*, vol. 7, no. 3, pp. 160-163, 1964.

[67] Daskalaki, S. and Birbas, T., "E±cient solutions for university timetabling problem through integer programming," *European Journal of Operational Research*, pp. 106-121, 2005.

[68] Daskalaki, S., Birbas, T. and Housos., "An integer programming formulation for a case study in university timetabling," *European Journal of Operational Research*, pp. 117-135, 2004.

[69] David, P., "A constraint-based approach for examination timetabling using local repair techniques," in *Lecture notes in computer science: Proceedings of the 2nd international conference on the practice and theory of automated timetabling, PATAT 1997*, Toronto Canada, 1998, pp. 169-186.

[70] David, P., "A constraint-based approach for examination timetabling using local repair techniques," in *Second International Conference on the Practice and Theory of Automated Timetabling (PATAT-97)*, Toronto, Ontario, Canada, 1997, pp. 132-145.

[71] De Smet, G., "ITC2007—examination track: Practice and theory of automated timetabling (Technical report)," 2008.

[72] Dechter, Rina, *Constraint Processing*.: Morgan Kaufmann, 2003, pp. 123-128.

[73] Dechter, R., "Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition," *Artificial Intelligence*, vol. 41, no. 3, pp. 273–312, 1990.

[74] Desroches, S., Laporte, G. and Rousseau, J.M., "Horex: A computer program for the construction of examination schedules," *INFOR*, vol. 16, pp. 294-298, 1978.

[75] Di Gaspero, L., "Recolour, shake and kick: A recipe for the examination timetabling problem," in *Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling*, pp. 404-407.

[76] Di Gaspero, L., "Recolour, shake and kick: A recipe for the examination timetabling problem," in *Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling*, 2002, pp. 404-407.

[77] Di Gaspero, L. and Schaerf, A., "A tabu search techniques for examination timetabling," *Practice and Theory of Automated Timetabling III*, vol. 2079, pp. 104 -117.

[78] Di Gaspero, L. and Schaerf, A., "A tabu search techniques for examination timetabling," *Practice and Theory of Automated Timetabling III*, vol. 2079, pp. 104 -117, 2003.

[79] Dimopoulou, M. and Miliotis, P., "Implementation of a University Course and Examination Timetabling System," *The European Journal of Operational Research*, vol. 130, pp. 202-213, 2001.

[80] Dimopoulou, M. and Miliotis., "An Automated Course Timetabling System developed in a distributed Environment: a Case Study," *European Journal of Operational Research*, vol. 153, pp. 136-148, 2004.

[81] Dorigo M., Maniezzo, V. and Colorni, A., "Ant System: Optimisation by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, vol. 26, no. 1, pp. 29-41, 1996.

[82] Dowsland, K. A., "Off-the-peg or made-to-measure? Timetabling and scheduling with SA and TS," *Practice and Theory of Automated Timetabling II*, vol. 1408, pp. 37-52, 1998.

[83] Dowsland, K. A., "Simulate annealing," in *Modern heuristics techniques for combinatorial problems*, C. Reeves, Ed.: McGraw Hill, 1995, ch. 2, pp. 20-69.

[84] Dueck, G., "New optimisation heuristics for the great deluge algorithm and the record-to-record travel," *Journal of Computational Physics*, vol. 104, pp. 86-92, 1993.

[85] Duong, T. A., and Lam, K. H., "Combining constraint programming and simulated annealing on university exam timetabling," in *Proceedings of the 2nd international conference in computer sciences, research, innovation & vision for the future, RIVF 2004*, Hanoi, Veitnam, 2004, pp. 205-210.

[86] Easton, K., Nemhauser, G. and Trick, M., "Sports scheduling," *The Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, 2004.

[87] Freuder, E.C., Wallace R.J., "Partial Constraint Satisfaction," *Artificial Intelligence*, vol. 58, pp. 22-70, 1992.

[88] George M. White, Bill S. Xie, Stevan Zonjic, "Using tabu search with longer-term memory and relaxation to create examination timetable," *European Jouranl of Operational Research*, vol. 153, no. 1, pp. 80-91, 2004.

[89] George T. B., Opalikhin V. and Chung. C. J., "Using an Evolution Strategy for a University Timetabling System with a Web Based Interface to Gather Real Student Data," in *Genetic and Evolutionary Computation Conference*, Chicago, USA, 2003.

[90] Gervet, C., "Large combinatorial optimization problem methodology for hybrid models and solutions," *JFPLC*, 1998.

[91] Gislen, L., B. Soderberg, C. Peterson, "Complex scheduling with Potts neural networks," *Neural Computation*, vol. 4, pp. 805-831, 1992.

[92] Gislen, L., Soderberg, B., Peterson, C., "Teachers and Classes with Neural Nets," *International Journal of Neural Systems*, vol. 1, pp. 167-168, 1989.

[93] Glover, F., "Future paths for integer programming and links to artificial intelligence," vol. 13, pp. 533-549, 1986.

[94] Glover, F. and Laguna, M., "Tabu search," 1997.

[95] Glover, F. W. and Kochenberger, G. A., *Handbook of Metaheuristics*.: Kluwer Academic Publishers, 2003.

[96] Glover, F., Laguna, M. and Marti, R., "Fundamentals of scatter search and path relinking Control and Cybernetics," vol. 29, no. 3, pp. 653-684, 2000.

[97] Gogos, E., Alefragis, C. & Housos, P., "Multi-staged algorithmic process for the solution of the examination timetabling Problem," *Practice and theory of automated timetabling (PATAT)*, 2008.

[98] Goldberg, D., "Genetic algorithms in search, optimisation, and machine learning," 1989.

[99] Gunawan A., Ming K., Poh K., "A Mathematical Programming Model for A Timetabling Problem," *World Congress in Computer Science, Computer Engineering and Applied Computing*, pp. 42-47, 2006.

[100] Hansen, M. P. and Vidal, R. V. V., "Solving a real-world problem using an evolving heuristically driven schedule builder," *Evolutionary Computing*, vol. 6, no. 1, pp. 61-80, 1998.

[101] Hansen, P., and Mladenovic, N., "Variable neighbourhood search: principles and applications," *European Journal of Operational Research*, vol. 130, pp. 449–467, 2001.

[102] Hatamlou, A. and Meybodi, M. R., "A Hybrid Search Algorithm for Solving Constraint Satisfaction Problems," in *World Academy of Science, Engineering and Technology*, 2007.

[103] Hentenryck, P. V., "Constraint satisfaction in logic programming," in *Logic programming series*. Cambridge: MIT Press, 1989.

[104] Hertz, A., "Tabu search for large scale timetabling problems," *European Journal of Operational Research*, vol. 54, pp. 39-47, 1991.

[105] Holland, J., "Adaptation in Natural and Artificial Systems," *Univeristy of Michigan Press, Ann Harbor*, 1975.

[106] IBM ILOG Optimization and Analytical Decision Support Solutions. (2008, Apr.) [Online]. http://www-01.ibm.com/software/websphere/products/optimization/

[107] International Timetable Competition. (2002) [Online]. http://www.idsia.ch/Files/ttcomp2002/

[108] Isaai, M. T. and Singh, M. G., "Hybrid applications of constraint satisfaction and metaheuristics to railway timetabling: a comparative Study," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 31, no. 1, pp. 87-95, 2001.

[109] Jussien N., and Lhomme O., "Local search with constraint propagation and conflict-based heuristics," *Artificial Intelligence*, vol. 139, no. 1, pp. 21-45, 2002.

[110] Karp, R. M., "Reducibility among combinatorial problems," *Complexity of Computer Computations*, pp. 85-104, 1972.

[111] Kendall, G. and Mohd Hussin, N., "A tabu search hyper-heuristic approach to the examination timetabling problem at The MARA University of Technology," *Practice and Theory of Automated Timetabling, Lecture Notes in Computer Science*, vol. 3616, pp. 270-293, 2005.

[112] Kingston, J. H., "A tiling algorithm for high school timetabling," in *Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling*, 2004, pp. 233-250.

[113] Kirkpatrick, S., Gelatt, C. D. and Vecchi, M. P., "Optimization by simulated annealiing," *Science*, vol. 220, pp. 671-830, 1983.

[114] Kocjan, W., "Dynamic Scheduling: State of the Art Report.," Technical Report T2002 2002.

[116] Kolodner, J., *Case-Based Reasoning*.: Morgan Kaufmann, 1993.

[117] Koza, J., "Genetic Programming: On the Programming of Computers by Means of Natural Selection," 1992.

[118] Landau, L. D. and Lifshitz, E. M., *Statistical Physics. 5*.: Oxford: Pergamon Press, 1980.

[119] Laporte, G. and Desroches, S, "Examination Timetabling by Computer," *Computers and Operations Research*, vol. 11, no. 4, pp. 351-360, 1984.

[120] Lennon, M. J. J., "Examination timetabling at the university of Auckland," *New Zealand Operational Research*, vol. 14, pp. 176-178, 1986.

[121] Leong, T. Y. and Yeong, W. Y., "A hierarchical decision support system for university examination scheduling," *Working paper, National University of Singapore*, 1990.

[122] Lotfi, V. and Cerveny, R., "A final-exam-scheduling package," *Journal of the Operational Research Society*, vol. 42, pp. 205-216, 1991.

[123] Lotfi, Vahidi and Cervenery, Robert , "A Final Exam Scheduling Package," *J. Opl Res. Soc.*, vol. 42, no. 3, pp. 205-216, 1991.

[124] Marti, R., Lourenco, H. and Laguna, M., "Assigning proctors to exams with scatter search," in *Computing Tools for Modeling, Optimization and Simulation*.: Kluwer Academic publishers, 2000, pp. 215-227.

[125] McCollum B. , McMullan P.J. , Parkes A. J. , Burke E.K. , Abdullah S., "An Extended Great Deluge Approach to the Examination Timetabling Problem," in *MISTA 2009, Multidisciplinary International Scheduling Conference: Theory and Applications*, Dublin, 2010.

[126] McCollum, B., Schaerf, A., Paechter, B., McMullan, P., Lewis, R., Parkes, A., Di Gaspero, L., Qu, R., & Burke, E. K., "Setting the research agenda in automated timetabling: The second international timetabling competition," *INFORMS Journal on Computing*, vol. 22, no. 1, pp. 120-130, 2010.

[127] McCollum, Barry and McMullan, Paul, "The Second International Timetabling Competition: Examination Timetabling Track," University of Nottingham, Queen's University, Notingham, Belfast, Technical Report: QUB/IEEE/Tech/ITC2007/Exam/v4.0/17 2007.

[128] Merlot, L. T. G., Boland, N., Hughes, B. D. and Stuckey, P. J., "A Hybrid Algorithm For The Examination Timetabling Problem," *Practice and Theory of Automated Timetabling IV, Lecture Notes in Computer Science*, vol. 2740, pp. 207-231, 2003.

[129] Metropolis, N., Rosenbluth A. W., Rosenbluth, M. N., Teller A. H. and Teller, E., "Equation of state calculations by fast computing machines," *Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087-1092, 1953.

[130] Michael R. Gary and David S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. SanFrancisco, CA: W.H. Freeman, 1979.

[131] Michalewicz, Z., *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd ed.: Springer-Verlag, 1996.

[132] Miner, S., S. Elmohamed, and H. W. Yau., "Optimizing Timetabling Solutions Using Graph Coloring," *NY : NPAC REU program, NPAC*, 1995.

[133] MirHassani, S. A., "A Computational Approach to Enhancing Course Timetabling with Integer Programming," 2005.

[134] Mitchell, M., *An introduction to genetic algorithms*. MIT, USA, 1996.

[135] Mladenovic, N. and Hansen, P., "Variable neighbourhood search," vol. 24, no. 11, pp. 1097–1100, 1997.

[136] Monfroy, E., "A Coordination-based Chaotic Iteration Algorithm for Constraint Propagation," in *Proceedings of the 2000 ACM Symposium on Applied Computing*, 2000, pp. 262–269.

[137] Monfroy E. and Rety J.H., "Chaotic iteration for distributed constraint propagation," in *14th ACM Symposium on Applied Computing*, 1999, pp. 19–24.

[138] Moscato, P., "Memetic algorithms: A short introduction," *New Ideas in Optimization*, pp. 219-234, 1999.

[139] Moscato, P., "On evolution, search, optimisation, genetic algorithms and martial arts: towards memetic algorithms," California Institute of Technology, 826, 1989.

[140] Müller, T., "ITC2007 solver description: A hybrid approach.," in *Proceedings of the 7th international conference on the practice and theory of automated timetabling*, Montréal, Canada, 2008.

[141] Muller T., Bartak R. and Rudov H., "Iterative Forward Search: Combining Local Search with Maintaining Arc Consistency and a Conflict-based Statistics," in *LSCS'04 - International Workshop on Local Search Techniques in Constraint Satisfaction*, 2004.

[142] Osman, I. H. and Kelly, J. P., "Metaheuristics: Theory and Application," 1996.

[143] Osman, I. H. and Laporte, G., "Metaheuristics: A bibliography," *Annals of Operations Research*, vol. 63, pp. 513-623, 1996.

[144] Pascal Van Hentenryck, Laurent Michel, and Yves Deville, *Numerica: a Modeling Language for Global Optimization*.: The MIT Press, 1997.

[145] Pesant G., Gendreau M., "A view of local search in constraint programming," in *Principles and Practice of Constraint Programming*, Berlin, 1996, pp. 353-366.

[146] Pillay, N., "A developmental approach to the examination timetabling problem (Technical report)," 2008.

[147] Prestwich, D., S., "Exploiting Relaxation in Local Search," *First International Workshop on Local Search Techniques in Constraint Satisfaction*, pp. 49-61, 2004.

[148] Prosser, P., "Hybrid algorithms for the constraint satisfaction problem," *Computer Intelligence*, vol. 9, no. 3, pp. 268–299, 1993.

[149] Qu, R., "Case-based reasoning for course timetabling problem," School of Computer Science and Information Technology, University of Nottingham, PhD Thesis 2002.

[150] Qu, R. and Burke, E. K., "Hybridizations within a graph-based hyper-heuristic framework for university timetabling problems," *Journal of the Operational Research Society*, vol. 60, pp. 1273-1285, 2009.

[151] Qu, R., Burke, E. K., McCollum, B., Merlot, L. T. G., and Lee, S. Y., "A survey of search methodologies and automated system development for examination timetabling," *Journal of Scheduling*, vol. 12, no. 1, pp. 55-90, 2009.

[152] Rechenberg, I., "Cybernetic solution path of an experimental problem," *Royal Aircraft Establishment*, vol. 1122, 1965.

[153] Reeves, C., *Modern Heuristics Techniques for Combinatorial Problems*.: McGraw Hill, 1995.

[155] Resende, M. G. C. and Ribeiro, C. C., "Greedy randomised adaptive search procedures Handbook of Metaheuristics," *Handbook of Metaheuristics*, pp. 219-249, 2003.

[156] Ribeiro Filho, G. and Lorena, L. A. N, "A constructive evolutionary approach to school timetabling," in *, Applications of Evolutionary Computing, Lecture Notes in Computer Science*.: Springer-Verlag, 2001, vol. 2037, pp. 130-139.

[157] Ribeiro, C. C. and Hansen, P., "Essays and Surveys in Metaheuristics," 2001.

[158] Richards E. T., Richards B., "Non-systematic search and learning: An empirical study," in *Conference on Principles and Practice of Constraint Programming*, Pisa, Italy, 1998, pp. 370-384.

[159] Roman Barták, Tomáš Müller, and Hana Rudová, "Minimal Perturbation Problem – A Formal View.," *Neural Network World*, vol. 13, no. 5, pp. 501-511, 2003.

[160] Romuald Debruyne and Christian Bessiere, "Some practicable filtering techniques for the constraint satisfaction problem," in *Proceedings of the Fifteenth International Conference on Artificial Intelligence (IJCAI'97)*, Nagoya, Japan, 1997.

[161] Ross, P., Corne, D., Fang, H., "Improving Evolutionary Timetabling with Delta Evaluation and Directed Mutation," *PPSN III*, 1994.

[162] Ross, P., Hart, E. and Corne, D., "Some observations about GA-based exam timetabling," *Practice and Theory of Automated Timetabling II*, vol. 1408, pp. 115-129, 1998.

[163] Ross, P., Hart, E. and Corne, D., "Some observations about GA-based exam timetabling," in *Practice and Theory of Automated Timetabling II, Lecture Notes in Computer Science*.: pringer-Verlag, 1998, vol. 1408, pp. 115-129.

[164] Rudov H. and Murray K., "University Course Timetabling with Soft Constraints," pp. 310-327, 2003.

[165] Schaerf, A., "A survey of automated timetabling," *Artificial Intelligence Review, Volume 13, Issue 2*, pp. 86-127, April 1999.

[166] Schaerf, A., "Combining local search and look-ahead for scheduling and constraint satisfaction problems," in *JCAI-97*, Nagoya, Japan, 1997, pp. 1254-1259.

[167] Schaerf, A. and Di Gaspero, L., "Local search techniques for educational timetabling problems," in *Proceeding of the 6th International Symposium on Operational Research*, Slovenia, 2001, pp. 13-23.

[168] Schaerf, A. and Schaerf, M., "Local search techniques for high school timetabling," in *Proceedings of the 1st Intl. Conference on the Practice and Theory of Automated Timetabling*, 1995, pp. 313-324.

[169] Schaerf, A. and Schaerf, M., "Local search techniques for high school timetabling," in *Proceedings of the 1st Intl. Conference on the Practice and Theory of Automated Timetabling*, 1995, pp. 313-323.

[170] Schank, R.C. and Abelson, R.P., *Scripts, plans, goals and understanding*. New Jersey, USA: Erlbaum, Hillsdale, 1977.

[171] Schiex T., Verfaillie G., "Nogood recording for static and dynamic constraint satisfaction problems," *Internet Journal, Artificial Intelligence Tools*, vol. 3, no. 2, pp. 187–207, 1994.

[172] Schimmelpfeng k., Helber S., "Application of a real-world university-course timetabling model solved by integer programming," 2006.

[173] Selim, S.M., "Split Vertices in Vertex colouring and their application in developping a solution to the faculty timetable problem," *The Computer Journal*, vol. 31, pp. 76-82, 1988.

[174] Selman B., Kautz H. A., and Cohen, "Noise strategies for improving local search," in *The 12th Nat. Conf. on Artificial Intelligence (AAAI-94)*, 1994, pp. 337–343.

[175] Shaw, P., "Using constraint programming and local search methods to solve vehicle routing problems," in *4th Conference on Principles and Practice of Constraint Programming*, Pisa, Italy, 1998, pp. 417-431.

[176] Tamura, Naoyuki, "Calc/Cream: Open Source Spreadsheet Front-End for Constraint Programming," *Lecture Notes in Computer Science*, vol. 4369, pp. 81-87, 2006.

[177] Tamura, Naoyuki. (2009, Feb.) Cream: Class Library for Constraint Programming in Java. [Online]. http://bach.istc.kobe-u.ac.jp/cream/

[178] Terashima-Marín, H., Ross, P.M. and Valenzuela-Rendón, "Evolution of constraint satisfaction strategies in examination timetabling," in *Proceedings of the Genetic and Evolutionary Conference*, 1999, pp. 635-643.

[179] Terashima-Marín, H., Ross, P.M. and Valenzuela-Rendón, M., "Clique-based crossover for solving the timetabling problem with GAs," in *Proceedings of the Congress on Evolutionary Computation*, 1999, pp. 1200-1206.

[180] The Message Passing Interface (MPI) standard. [Online]. http://www.mcs.anl.gov/research/projects/mpi/

[181] Thompson, J. M. and Dowsland, K. A., "A robust simulated annealing based examination timetabling system," *Computers Operational Research*, vol. 25, no. 7/8, pp. 637-648, 1998.

[182] Thompson, J. M. and Dowsland, K. A., "General Cooling Schedules for a Simulated Anealing Based Timetabling System," in *Practice and Theory of Automated Timetabling, Lecture Notes in Computer Science*.: Springe-Verlag, 1996, vol. 1153, pp. 345-363.

[183] Timothy, A.R., "A Study of university timetabling that blends graph coloring with the satisfaction of various essential and preferential conditions," Ph.D. Thesis 2004.

[184] Trick, M.A., "A schedule-then-break approach to sports timetabling," in *The Practice and Theory of Automated Timetabling III*, vol. 2079, 2001, pp. 242-253.

[185] Valdes R.A., Crespo E., and Tamarit J.M., "Design and implementation of a course scheduling system using Tabu Search," *European Journal of Operational Research*, vol. 37, pp. 512-523, 2002.

[186] Van Hentenryck, Pascal and Saraswat, Vijay, "Constraint Programming: Strategic Directions," *Constraints: An International Journal*, vol. 2, pp. 7-33, 1997.

[187] Verfaillie G., Schiex T., "Solution reuse in dynamic constraint satisfaction problems," in *AAAI-94*, Seatle, WA, USA, 1994, pp. 307–312.

[188] Voß, S., "Metaheuristics: The State of the Art," *Local Search for Planning and Scheduling*, pp. 1-23, 2001.

[189] Voß, S., Martello, R., Osman, I. H. and Roucairol, C., "Metaheuristics: Advances and Trends in Local Search Paradigms Optimization," 1999.

[190] Weare, R. F, "Automated examination timetabling," Depratment of Computer Science, University of Nottingham, UK, Ph.D. Thesis 1995.

[191] Welsh, D. J. A. and Powell, M. B., "An upper bound for the chromatic number of graph and its application to timetabling problems," *The Computer Journal*, vol. 10, no. 1, pp. 360-364, 1967.

[192] Werra, D., "An Introduction to Timetabling," *European Journal of Operations Research*, vol. 19, pp. 151-162, 1985.

[193] White, G. M. and Chan, P. W., "Towards the construction of optimal examination timetables," *INFOR*, vol. 17, pp. 219-229, 1979.

[194] White, G. M. and Xie, B. S., "Examination timetables and tabu search with longer term memory," *Practice and Theory of Automated Timetabling III*, vol. 2079, pp. 85-103, 2001.

[195] White, G.M., Xie, B.S. and Zonjic, S., "Using Tabu Search with Longer-Term Memory and Relaxation to Create Examination Timetables," *European Journal of Operational Research*, vol. 153, pp. 80-91, 2004.

[196] Wikipedia (2012). [Online]. http://en.wikipedia.org/wiki/Red-black_tree

[197] Wikipedia (2011, Dec.), the free encyclopedia. [Online]. http://en.wikipedia.org/wiki/Great_Deluge_algorithm

[198] Williams, H. P., *Model building in mathematical programming*. Chichester: Wiley, 1999.

[199] Wood, D. C. A., "Technique for colouring a graph applicable to large scale timetabling problems," *The Computer Journal*, vol. 12, pp. 317-319, 1969.

[200] Wren, A., "Scheduling, Timetabling and Rostering - A Special Relationship.," *Practice and Theory of Automated Timetabling.*, pp. 46-75, 1996.

[201] Wren, A., "Scheduling, Timetabling and Rostering - A Special Relationship.," *Practice and Theory of Automated Timetabling.*, pp. 46-75, 1996.

[202] Yang, Y. and Petrovic, S., "A novel similarity measure for heuristic selection in examination timetabling," in *Lecture notes in computer science: Vol. 3616 Proceedings of the 5th international conference on the practice and theory of automated timetabling, PATAT 2004*, Pittsburg, PA, USA, 2005, pp. 377-396.

[203] Yokoo, Makoto, "Foundations of Cooperation in Multi-Agent Systems," in *Distributed Constraint Satisfaction*.: Springer-Verlag, 2001.

[204] Yokoo, M., "Weak-commitment search for solving constraint satisfaction problems," in *AAAI-94*, Seattle, WA, USA, 1994, pp. 313-318.

Appendix A

Solver Model Data Structure (Technical Review)

In computer science, there are always trade-offs when it comes to efficiency and optimizations. Nothing comes for free, and in order to have the ability to perform operations in a smooth way like insertion, indexed search and lookups, items deletion, we need to have some sort of data structure that has the best performance which will support this ability. Indeed, selecting the right data structure for any constraint satisfaction problem or optimization problem, especially problems with large sets of variables and values, is always vital in determining the efficiency of any solver. Because we use Microsoft .NET framework as a backbone we thought it is worth studying the different types of collections in .NET in terms of time complexity.

It is worth noting that most collections in .NET are implemented as generics. In other words, we can refer to a class, where we don't force it to be related to any specific type, but it can still perform work with it in a Type-Safe manner. A perfect example of where we would need Generics is in dealing with collections of items (integers, strings, user defined classes… etc.). We can create a generic collection that can handle any Type in a generic and Type-Safe manner.

This appendix provides a full overview study of some of these collections that we considered using in IDS Solver along with a benchmarking and reasoning on why and when we used some and not used others.

Prior to do any coding in the developed IDS solver, we decided to do a quick study for the different data structure types offered by Microsft.net framework (The IDS Solver's main backbone framework). In this chapter, we provide a short summary and benchmarking on efficiency and performance of .Net collections. We also provide a brief reasoning on why we decided to use some collections and not others. Selecting the right data structure for any application is definitely will help its performance on the long run.

A.1. Collections in .NET framwork

A.1.1. List<TValue>

This collection seems to be the fallback if the developer is confused over what to use, and with a good reason (TValue in the any .NET collection represents any Type). It is unsorted and items can be added, removed, or inserted at a specific index. It also maintains support to sorting on demand. Additionally, it has indexed access, which means that items can be accessed directly using a numeric index. Likewise, a list range can be added at once. It also can perform binary searches, perform sequential searches, built in sorting, get the count of items, check for items within it, etc… It is considered as the Swiss Army knife of collections.

List<T> is internally implemented as a simple array. As more items added to the list, an entirely new array is allocated if the array fills up. Also, all items in the current array are copied over into the new array and then the new item is added. Most of the operations are really straight forward and we can see why List<T> is the generic fallback for any list of unknown length, it provides good performance across a very wide variety of operations.

A.1.2. HashTable

Hash tables are part of a collection family called associative maps, or associative arrays. Hash tables usually map a collection of keys to a collection of values. Associative maps themselves are part of a wider family of data structures known in computer science as containers. In a hash tables, a fast keyed array are created where the elements in the array can all be located using a key value. Accessing hash table collection is done using a key, a calculation, named a hashing algorithm. Some documentation refers to this operation as a O(1) lookup rather than an array's O(n) lookup.

A.1.3. SortedList<TKey, TValue>

SortedList is very similar to List, with one noticeable difference, it is sorted. An item is added as means of provided sort key and a value, which are both stored. On inserting an item, a binary search is performed across the list in order to find the right place for it, and then the item is inserted into the array. With the objective

of doing this, the entire array has to be shifted in order to make room for the new item. For this reason, inserting items into the SortedList is considered as an expensive operation. But because the list is maintained and sorted each time an item is inserted, added or removed. On the other hand, lookups are considered very cheap. Both, a key and a value are needed in most operations and since lookups are relatively so cheap, if you need a list which will have heavy lookups with a lower number of inserts, then the SortedList is thought to be the best alternative.

A.1.4. Dictionary<TKey, TValue>

Dictionary class in its essence is an implementation of a dictionary data structure. It also referred to as an associative array or a map. Its operations look very comparable to a SortedList, since it is operable using both a key and a value, but in reality they have very diverse performance features due to infrastructure implementation.

There is a specific reason for this diversion which is while a SortedList performs an insertions and lookups using a binary search mechanism, the Dictionary class essentially uses a hash table as its backing store. This is, in turn, means that the data stored inside of the dictionary isn't sorted, but rather it is looked up by a hash value. And hence the keys must also be unique. This makes looking up individual

items enormously fast by hashing them, and then looking up their hash within the hash table.

That is why finding a collection of values would require iterating over all dictionary values looking for searched values. It also means that the data stored in the Dictionary, when iterated over, comes back in non-determinant order usually in inserted order and not in a sorted order.

A.1.5. SortedDictionary<TKey, TValue>

SortedDictionary is not one of the favourite collections in .NET as most users might not even know what differentiates a SortedDictionary from a Dictionary from a SortedList. SortedDictionary is actually quite a different collection. Internally its structure is implemented as a Red-black tree [196]. Red-black trees are a much more complicated data structure than other collections' internal implementation. The differences can be summarized in two points. First is that they have great worst-case operations. Second, which is the biggest difference between this and a SortedList, is that the SortedDictionary does not permit indexed access. It does however have more rapidly add, insertion, and deletion opertions. Also, it consumes more memory than SortedList because of the usage of tree structure in which the data is stored.

A.1.6. SortedSet<T>

SortedSet is essentially the backbone data structure for the SortedDictionary data type with one major difference. Unlike the SortedDictionary, there are only values and no keys. As it is the backbone for the SortedDictionary it has similar performance features.

In computer science, the SortedSet represents an abstract data structure called set. It is an abstract data structure because it does not involve any particular implementation. A set, fundamentally, is a collection that contains unordered unique values. This implementation uses a Red-black tree, which is a sorted data structure, in order to sustain the rules of the set. Theoretically a set does not have any order, however, this is a sorted set, and hence iterating over all items within the set, they will be in sorted order.

A.1.7. LinkedList<T>

LinkedList class is an implementation of the well-known double linked list data structure. A double linked list is implemented using a simple data structure involving of a cluster of nodes with pointers to the previous and next node.

The .Net Framework implementation, moreover point the next and previous pointers on the first and last node to each other. As a result of the way that this data structure works, it is assumed on of the cheapest collections to add, insert

and delete items, assuming that you are already in the place in the list where the data needs to be inserted or deleted.

Furthermore, since pointers only exist between nodes, there is no indexed access and in order to find items in the list, each item needs to be iterated through. The LinkedList is a simple implemented data structure, but it is an outstanding data structure if you have a list which has a substantial amount of insertion and deletion within the list. It is particularly useful if you are inserting ranges of data into the middle of a list.

A.1.8. Queue<T>

The Queue implements a queue data structure which serves a special purpose. A queue is also called “FIFO” (First In First Out) collection with has two main operations: Enqueue and Dequeue. Items are positioned into one end of the queue, and then are taken off the other end.

The Queue is categorically only useful if you need the exact flow that its data structure delivers which is to place items into the queue at one end and then to be able to take them off and process them in this specific order.

A.1.9. Stack<T>

Similar to queue, the Stack serves another special purpose data structure which implements another well-known computer science stack data structure. A stack is

also called “LIFO” (Last In First Out) collection with two operations: Push and Pop. Items are placed on the top of the collection and then are taken off the top.

A.1.10. HashSet<T>

Similar to HashTables, HashSet is a collection that implements set abstract data structure that uses hashes in order to maintain a unique set of items. Internally the items are put in storage of an array, which, as known, requires size expansion when the array reaches its current initiated size. Because hashes are used as a backbone collection, the data is unsorted, which means that searching for an item is $O(N)$ operation since all data would need to be examined during the search. On the other hand, hashes are very fast with insertions and lookups and so accomplishing any set operation such as unions or intersects is considered faster on a HashSet than a SortedSet. As indicated above, this collection has one drawback; inserting can cause the array to be expanded, which would cause the operation to be very expensive.

Since both HashSet and SortedSet implement the same interface which is named ISet, the single genuine purpose to use SortedSet over HashSet is that the data, which we need to act on, cannot be easily hashed. Otherwise, HashSet will probably be the faster of the two sets to use, in addition to its unique feature of using less memory.

A.2. Time Complexity

The following table illustrates the time complexity of the main operations that can be done on any .NET collection.

Collection/ Time Complexity	Add	Delete	Insert	Lookup	Sorted/Indexed Lookup
List<T> , List	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(log_n)$
HashTable	$O(1)$	$O(1)$	n/a	$O(n)$	n/a
SortedList<T>, SortedList	$O(n)$	$O(n)$	$O(n)$	$O(log_n)$	$O(log_n)$
Dictionary<TKey, TValue> Dictionary	$O(1)$	$O(1)$	n/a	$O(n)$	O(1)
SortedDictionary<TKey, TValue>, SortedDictionary	$O(log_n)$	$O(log_n)$	$O(log_n)$	n/a	$O(log_n)$
LinkedList<T>	$O(1)$	$O(1)$	$O(1)$	$O(n)$	n/a
SortedSet<T>	$O(log_n)$	$O(log_n)$	$O(log_n)$	N/A	$O(log_n)$
HashSet<T>	$O(1)$	$O(1)$	n/a	$O(n)$	$O(1)$
Queue<T>	$O(1)$	$O(1)$	n/a	$O(n)$	n/a
Stack<T>	$O(1)$	$O(1)$	n/a	$O(n)$	n/a

Table 12 - .NET Collections Time Complexity

A.3. Collections Benchmarking

Prior to deciding which collection to use, we filtered out some of the collections mentioned in the section above. And we only considered examining and benchmarking the ones that we well thought out using in IDS Solver. The benchmarking only involves four main operations and memory consumption. The four operations are insert, remove, lookup and iterating through (for each). The

experiment was done on a machine with i7 Intel processor with 4GB memory. We used a collection of integers in the whole experiment. We used a range of variables starting from 5 variables and ending with 10,000 variables. Then after collecting results, we interpolated results to see the full picture.

Based on the results we got below, we made the decision to use Sorted Generic Lists for all CSP variable, value and constraints collections. There are two reasons for that. First, we only need to build them once at the beginning of building the model. They do not consume a lot of memory as other types of collections. In fact, they are the least memory consuming collection. The second reason is that after building any of the CSP collections, we only need to do lookups which a generic list is good at and one of the fastest collections for that matter.

For other type of operations as building tabu lists and assigned and unassigned variables, we chose to use hash sets as they provide the fastest performance for inserting and removing items. Lists will be too expensive for these types of operations. In reality any collections that implements hashing as its infrastructure would be fine for this role. But we chose hash sets as they allow for duplication of items as most sets do.

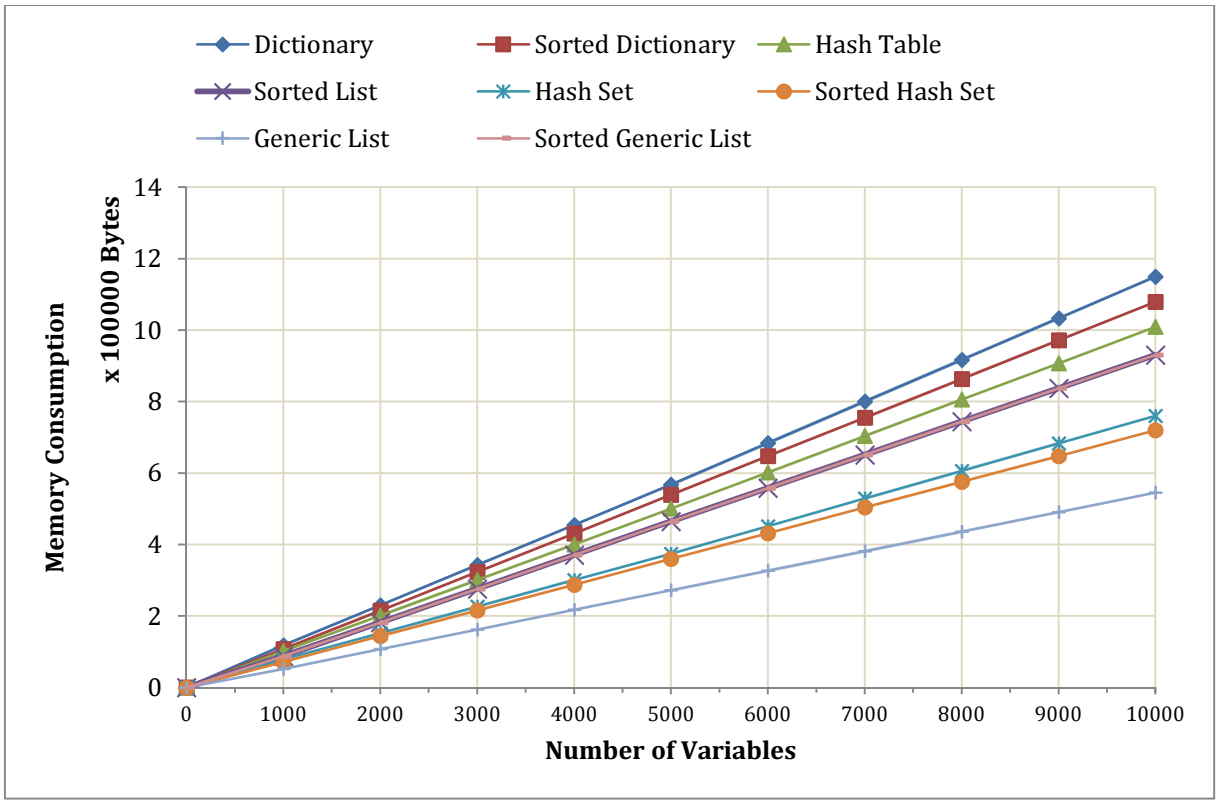


Figure 32 - Memory consumption in .NET Collections

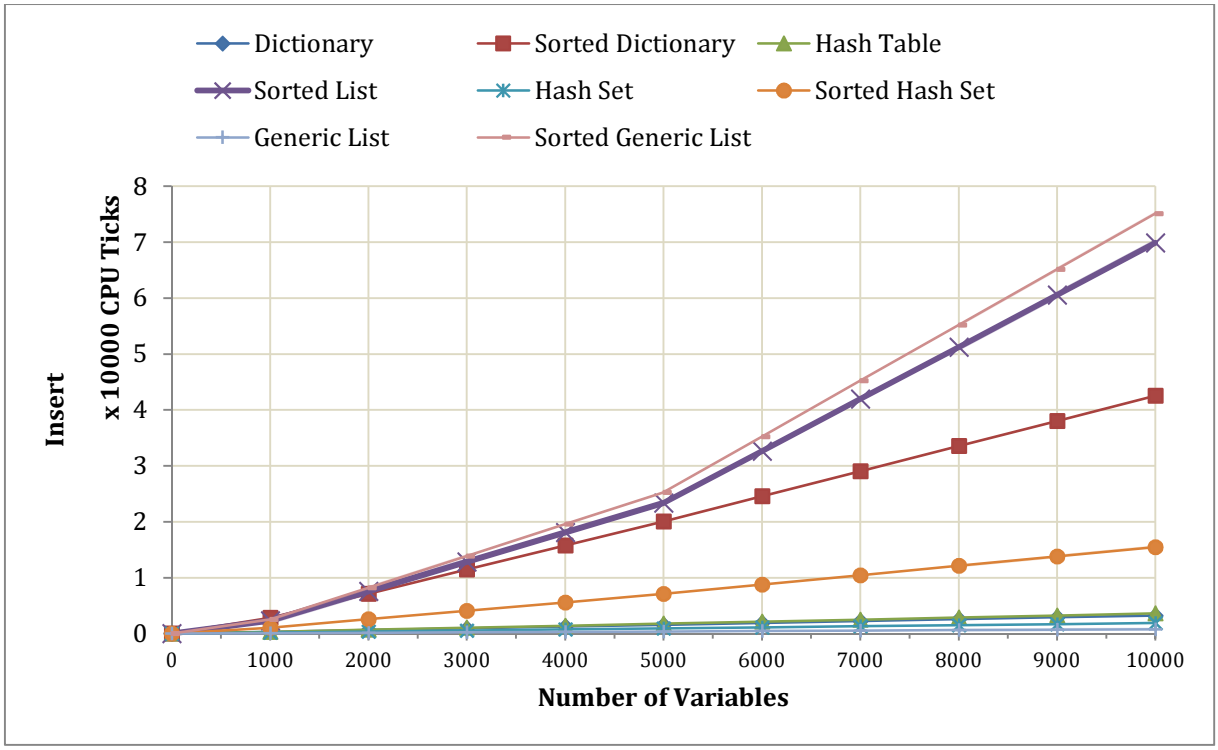


Figure 33 - Insert Operation in .NET Collections

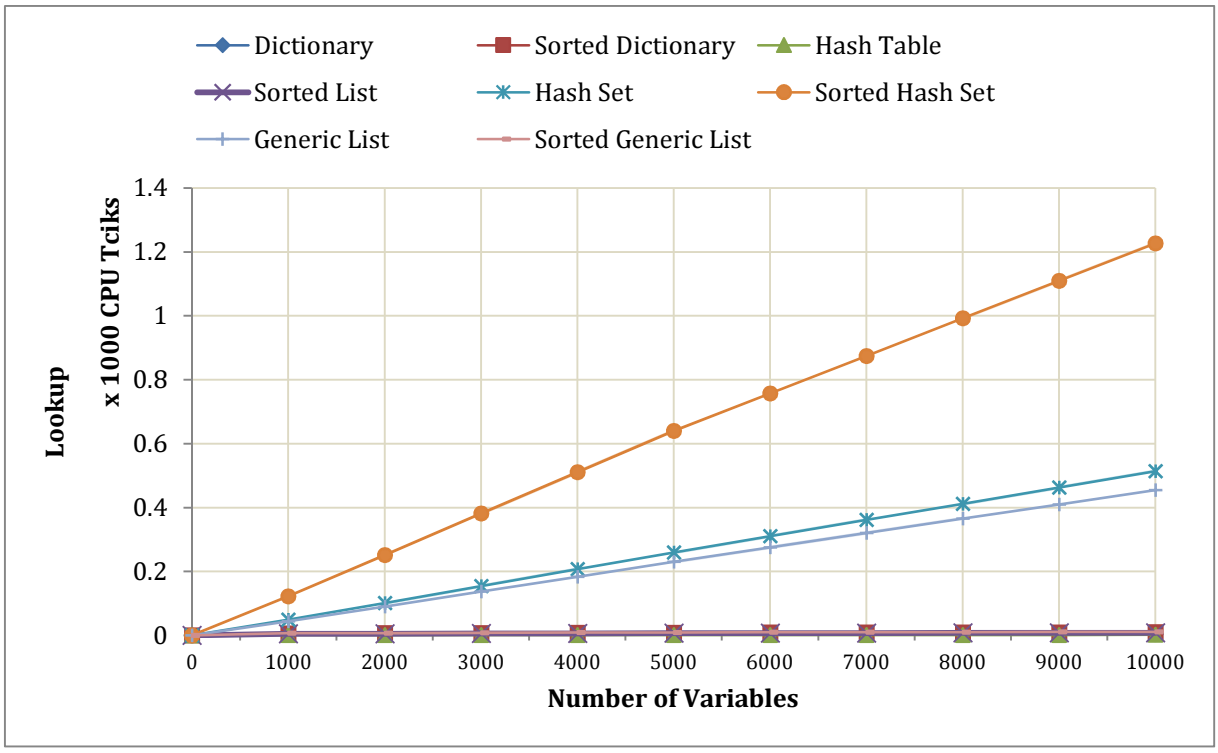


Figure 34 - Lookup operation in .NET Collections

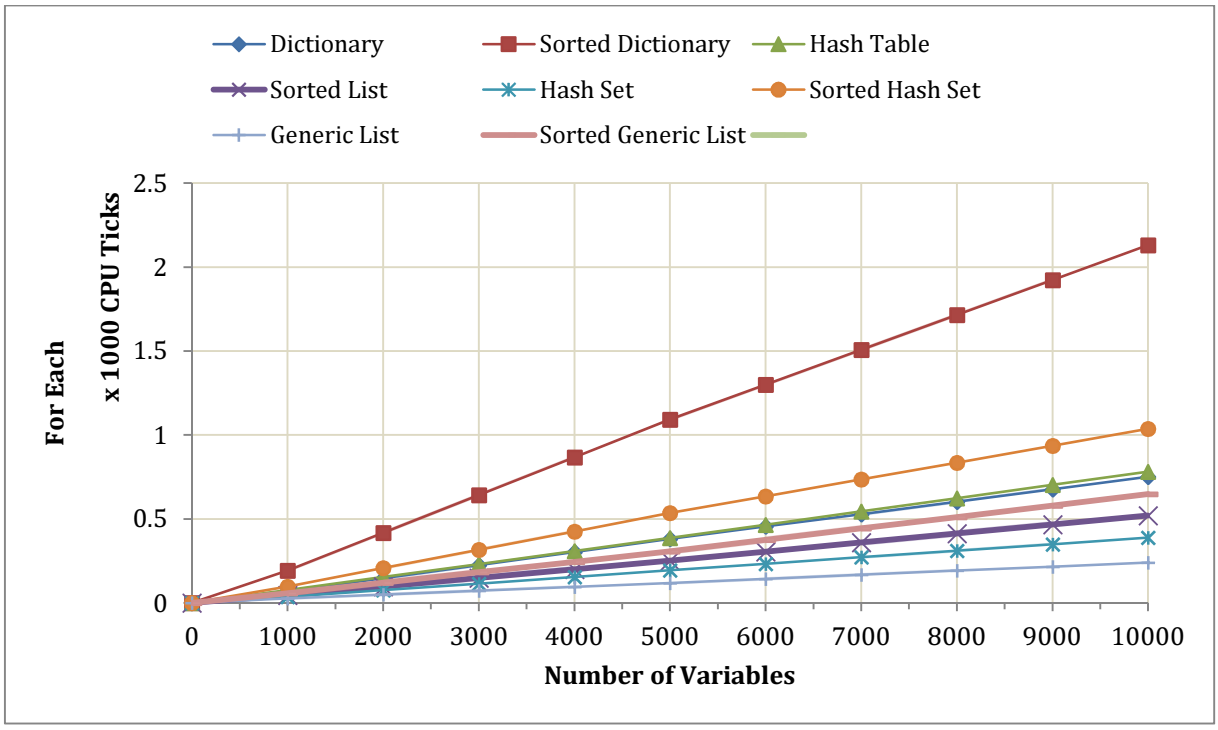


Figure 35 - For Each operation in .NET Collections

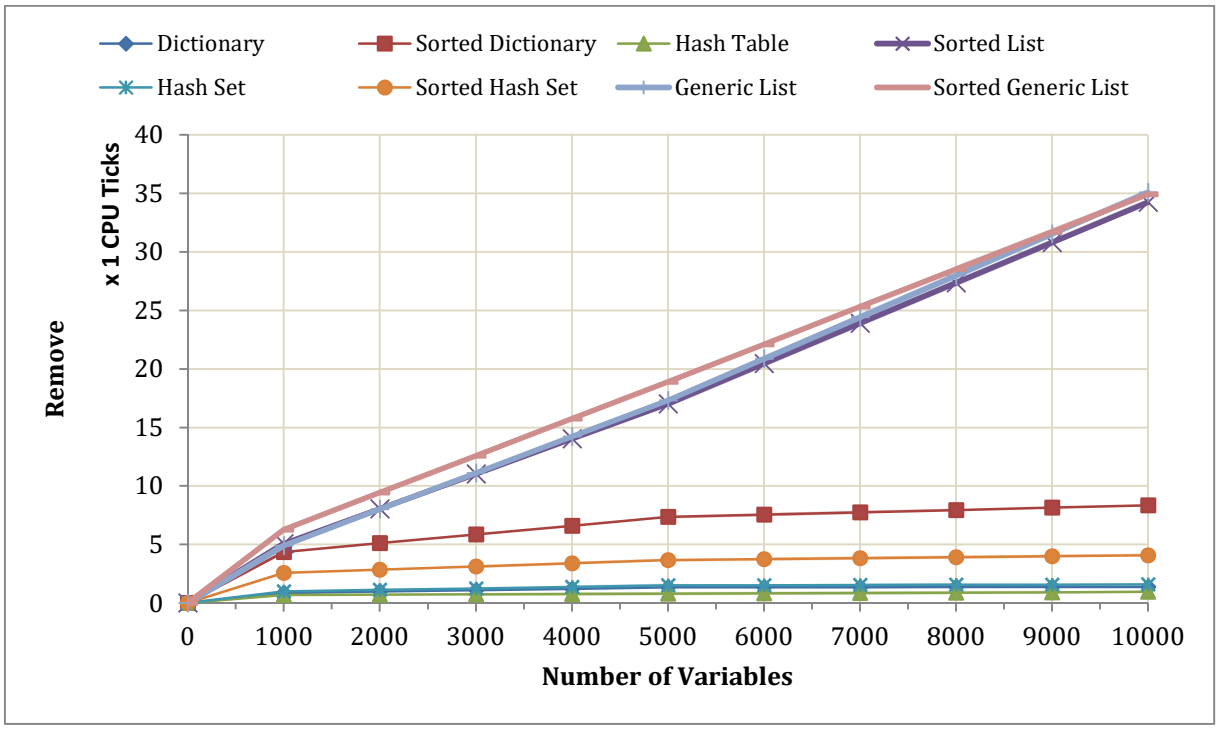


Figure 36 - Remove Operation in .NET Collections

Appendix B

Exam Timetabling Benchmarking Datasets

Generally, most of the literature on examination timetabling can be divided into two main categories: The first, which is less used in literature, where it only focuses on solving examination timetabling problems from practice by involving real life problems usually where the researcher is doing her study. The second category concentrates on improving the results of some of the well-known benchmarking. In this appendix we detail two of the well-known benchmarking datasets in literature regarding Exam Timetabling Problem. The first is the Toronto Benchmarking datasets and it is relatively old and has been used since 1996. The second was introduced in the second International Timetabling Competition 2007 and has been since used in many research papers.

B.1. Toronto Benchmarking Datasets

Toronto Benchmarking Dataset is one of the most studied examinations timetabling benchmark datasets and was introduced by Carter et al. [52]. The benchmark dataset consists of 13 basic real world examination time tabling problems obtained from different universities around the world. The benchmark datasets are also known as the incapacitated Toronto benchmarks. Unlike real life examination timetabling problems and strangely enough, these benchmark instances do not take

into consideration rooms and lecturers. It assumes that rooms have unlimited seating places.

The Toronto benchmarking datasets were considered as the standard test dataset for examination timetabling problem up until the second International Timetabling Competition in 2007 introduced new benchmarking datasets [62]. Toronto benchmarking datasets have two main objectives; constructing conflict-free examination timetables for every student, and spreading out exams evenly in a way that every student should have enough time to study for the next exam. Different versions of the Toronto instances are available but the most commonly applied version in literature is the one introduced by Qu et al. in 2009 [151].

The detailed properties of the 13 Toronto benchmark instances are summarized in Table 13. The conflict density value is calculated using the ratios of the total number of non-zero elements in the conflict matrix to the total number of elements of the conflict matrix. The conflict matrix is calculated as follows. For every exam, a new entry is added to the conflict matrix that contains the number of students that share that exam; zero if there are none. The conflict density value gives a relatively good sign of the number of conflicts between the individual exams.

Due to the over-work of research that was done on Toronto benchmarking dataset, we decided not to use it in our experiments as our benchmarking basis dataset. Rather, we chose to use ITC 2007 Exam Timetabling Track Benchmarking Dataset as

they relatively new dataset that still not excusably studied. We will go through in more detail in the next few sections later on in this chapter.

Instance	# of students	# exams	# enrolments	Conflict density	# time slots
Car 91	16,925	682	56,877	13%	35
Car 92	18,419	543	55,522	14%	32
Ear83	1,125	190	8,109	27%	24
Hec92	2,823	81	10,632	42%	18
Kfu93	5,349	461	25,113	06%	20
Lse91	2,726	381	10,918	06%	18
Pur93	30,032	2,419	120,681	03%	42
Rye92	11,483	486	45,051	07%	23
Sta83	611	139	5,751	14%	13
Tre92	4,360	261	14,901	18%	23
Uta92	21,266	622	58,979	13%	35
Ute92	2,749	184	11,793	08%	10
Yor83	941	181	6,034	29%	21

Table 13 - Toronto's 13 Benchmarking Datasets

B.2. International Timetabling Competition Benchmarking Datasets

Based on the success of the First International Timetabling Competition in 2002 [107], the second competition was announced in 2007 to invite researchers to develop and experiment their research and techniques in solving timetabling problems. Additionally, the competition had the objective of gaining more interest in the scheduling and timetabling research area by providing different and heterogeneous formulations of the timetabling problems that are faced within educational institutions every year based on "real world" problems. The competition, as a result, aims to help in narrowing the existing gap that exists between research and practice in regard to timetabling and scheduling problems.

In this competition, three tracks were introduced along with a number of associated benchmark datasets. The three tracks are examination timetabling (Exam TT), post-enrolment course timetabling (PostEnroll CTT) and curriculum-based course timetabling (Curriculum CTT). The examination timetabling track of the competition uses a more realistic model of the problem in terms of data, constraints and evaluation. All datasets used as part of this competition are taken from real institutions and have been made anonymous for the purpose of competition use. The benchmark dataset consists of 12 basic real world examination time tabling problems obtained from anonymous different universities around the world. The detailed properties of the 12 benchmark instances are summarized in Table 14.

Instance #	# of students	# of exams	# of rooms	Period & Room Hard Constraints	Constraints density	# of time slots
1	7,891	607	7	12	05.04%	54
2	12,743	870	49	14	01.17%	40
3	16,439	934	48	184	02.62%	36
4	5,045	273	1	40	14.94%	21
5	9,253	1,018	3	27	00.87%	42
6	7,909	242	8	23	06.13%	16
7	14,676	1,096	15	28	01.93%	80
8	7,718	598	8	21	04.54%	80
9	655	169	3	10	07.79%	25
10	1,577	214	48	58	04.95%	32
11	16,439	934	40	185	02.62%	26
12	1,653	78	50	16	18.21%	12

Table 14 - ITC 2007 Exam Track Benchmarking Datasets

Index

R

S

T

U