Event Scheduling:
An Ant-Inspired Hybrid Approach

By
Naveed Ejaz
2003-NUST-MS PhD-CSE-218

Submitted to the Department of Computer Engineering
In fulfillment of the requirements for the degree of

Master of Science
In
Computer Software Engineering

Thesis Supervisor
Dr. Muhammad Younus Javed
PhD (CS), UK

College of Electrical & Mechanical Engineering
National University of Sciences & Technology
2009

ACKNOWLEDGMENTS

All praise to Allah (The Almighty) who enlightened me with the requisite knowledge to accomplish this task.

I extend my deepest gratitude to all the people whose help and support enabled me achieve my goal. I am thankful to my parents for being a constant source of encouragement, to my wife for her patience and support, and to my brother, Khalid for his assistance and useful suggestions.

I thank the faculty members of Computer Engineering Department for their assistance and cooperation, especially Dr. Farooque Azam, Dr. Shaleeza Sohail and Dr. Ghalib Asadullah Shah.

Special thanks to my thesis supervisor Dr. Muhammad Younus Javed who has been a constant source of inspiration for me. Without his guidance, encouragement and support, this task would not have been possible.

.

Dedicated to my parents for their love, trust and care!

ABSTRACT

In the present day's fast-paced world, there is a great emphasis on organized and efficient use of resources and scheduling has become an important part of daily life; be it work, education, transport or entertainment. In many real-world cases, particularly where resources are not in abundance, and domain-specific requirements are complex, the construction of usable and effective schedules can be a very challenging task. Due to its importance and complexities involved in its construction, automation of scheduling is an imperative task for every sizable organization, to enable it to make the most out of its time and resources.

Event scheduling is a combinational optimization problem which belongs to a class of NP-complete problems along with other 'difficult-to-solve' problems like Traveling Salesman Problem, Bin-packing and Graph-coloring. In these problems, only surety to find best solution is by checking all the possible solutions using brute-force/exhaustive search, which is not practically possible due to very high computational costs. Being an NP-Complete problem, a time-bound solution for Scheduling problems can not be guaranteed by any of the algorithms. Therefore, new ideas and approaches for the solution provide new opportunities towards more complete and better working algorithms. In addition, different scenarios have different constraint-sets, which need different approaches towards solution; therefore, devising a general framework which can cater for different scenarios may be helpful in many application areas.

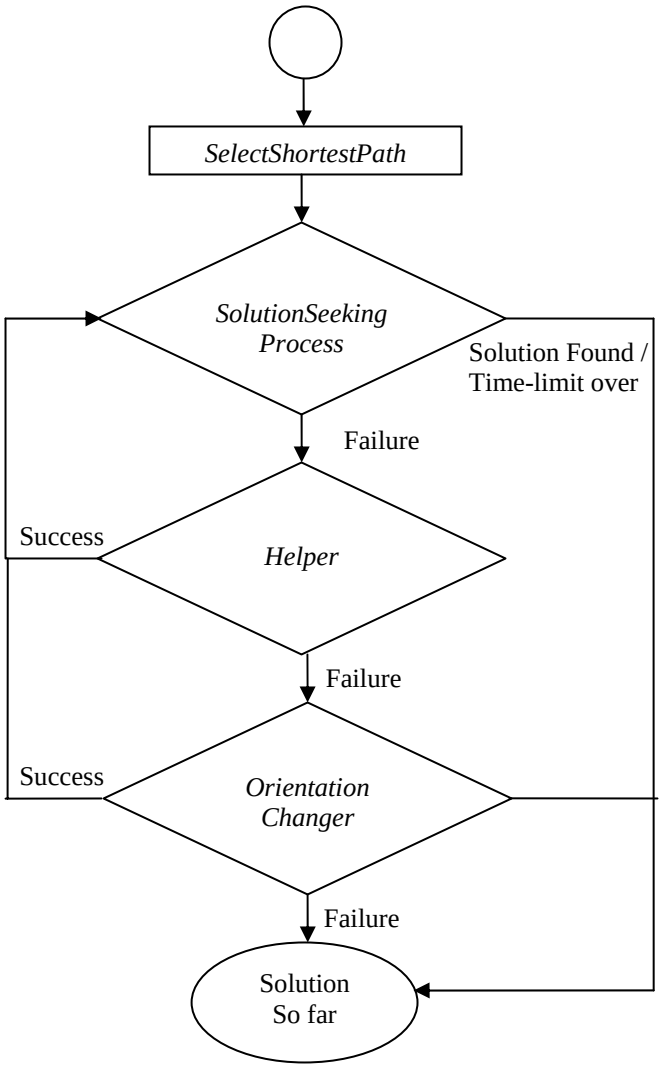
In this thesis, a hybrid two-stage framework has been presented. The approach is inspired by the mutual-aid and persistent/die-hard behavior of ants exhibited when faced with difficult scenario while collecting food, thus named "Die-Hard Co-Operative Ant Behavior Approach" (DCABA). An initial assignment of events is obtained with the help of a set of heuristics and it is evolved by searching promising areas of search-space by finding the problematic events instead of random search. The search space is limited by defining some more heuristics. In the first stage, a feasible solution is constructed and in the second stage, optimizer functions improve quality of the

obtained solution. Many different heuristics and techniques may be used within this framework. The approach has been applied on a set of University Course Scheduling Instances and promising results have been obtained. This approach may also be used for the solution of job shop scheduling, traveling salesman problem and vehicle route scheduling problem.

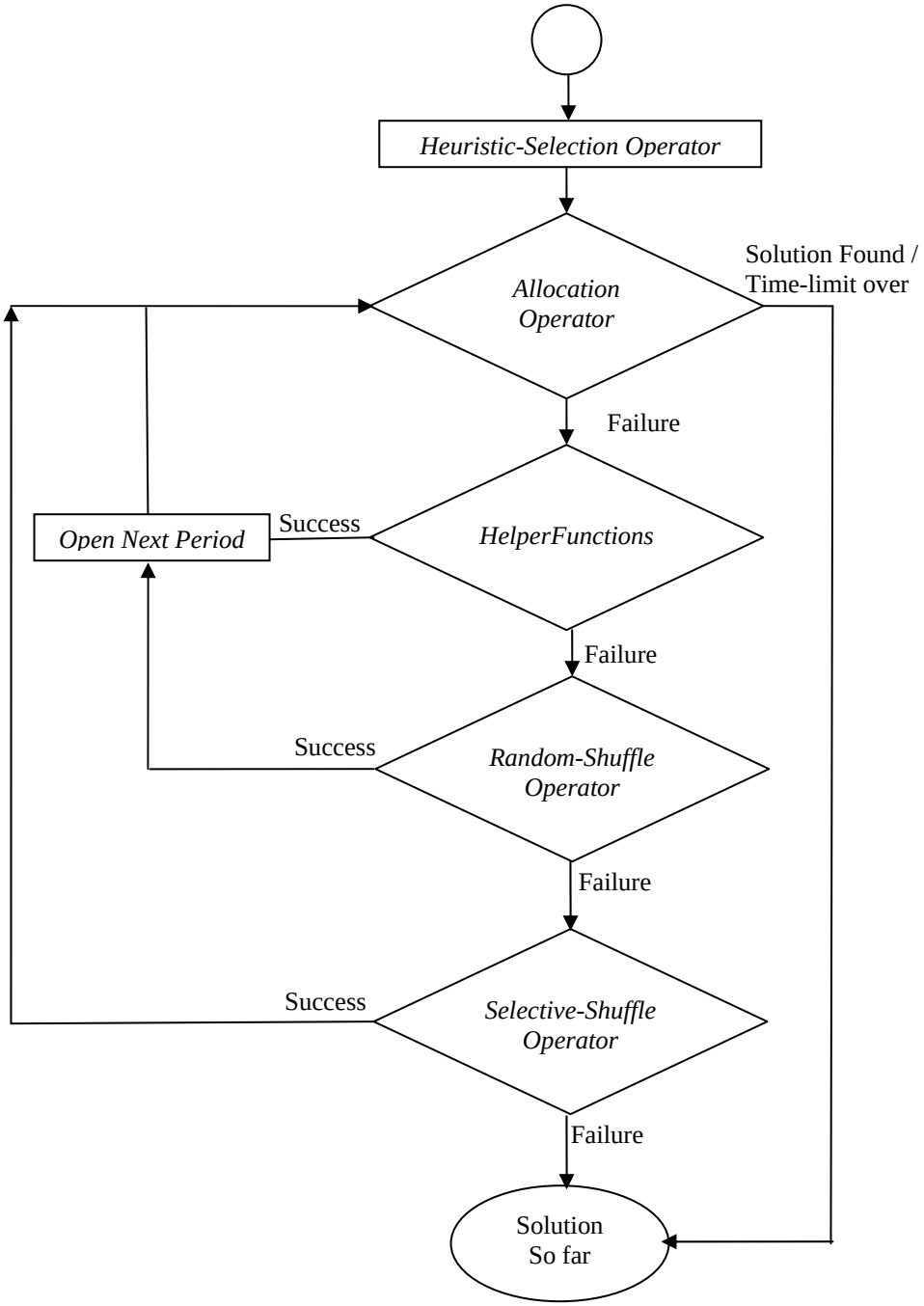
This research work is useful for a host of scenarios where automation of scheduling can help improve performance, efficiency and time management.

TABLE OF CONTENTS

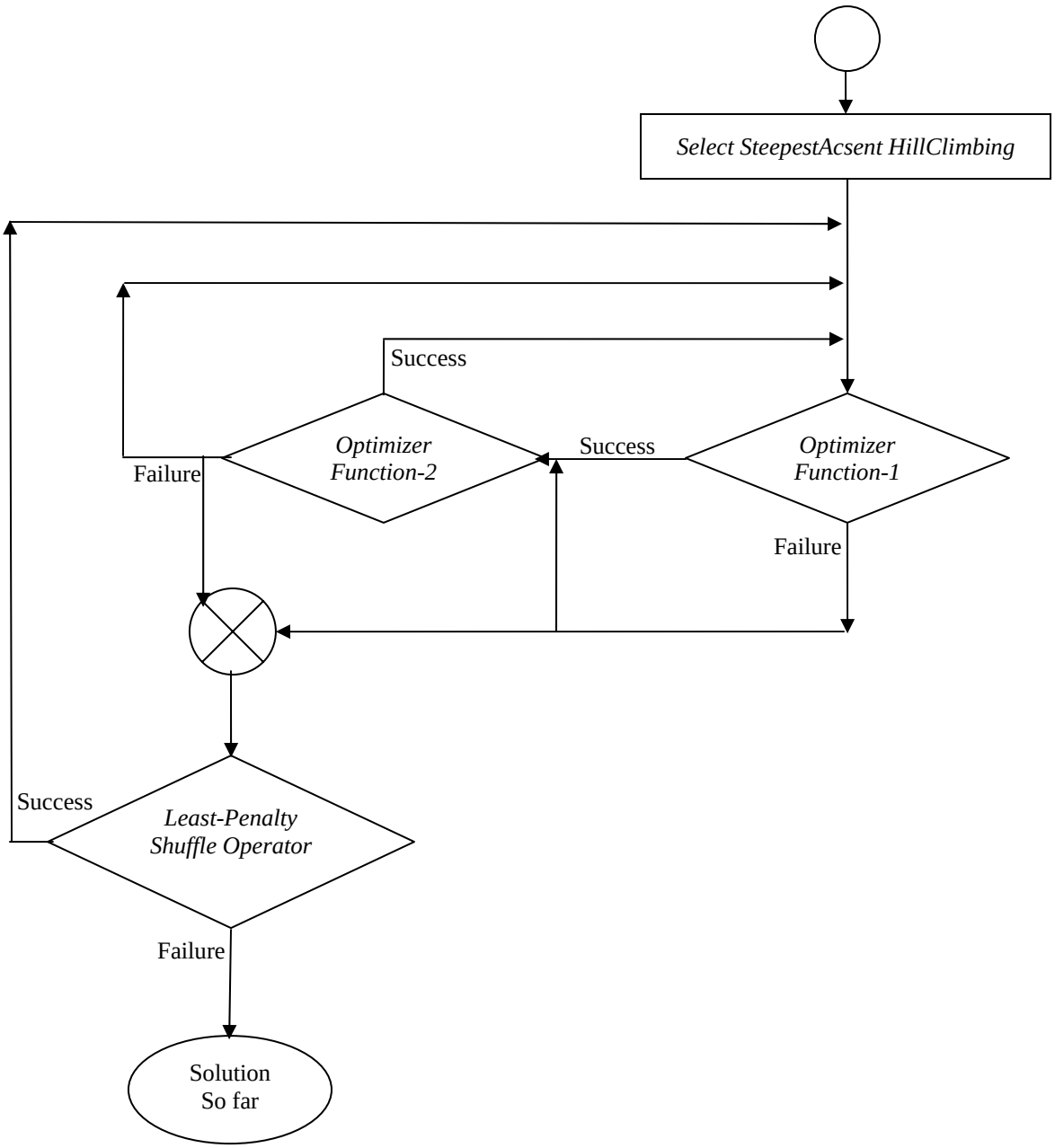
B.1 DCABA Model for Event Scheduling



B.2 DCABA Model for Allocation Phase



B.3 DCABA Model for Optimization Phase



Chapter 1: Introduction

1.1 Background and Motivation

Scheduling has become an important part of today's organized and modern world. Schedules, timetables and agenda are everywhere in many areas of daily life. These schedules often need to be updated and re-made on a regular basis. Given this fact and the importance of schedules in the daily lives of the people, automation of scheduling problems attracts a lot of interest and effort from researchers. In many cases, particularly where there is an emphasis on efficient use of resources, the problem of constructing usable and effective schedules can be a demanding task.

From a researcher's perspective, generally all variants of scheduling problems belong to the class of NP-complete problems, as will be discussed in detail later. So there is no known deterministic polynomially bounded algorithm for solving them. Secondly, different scheduling problems also become complicated due to the needs and priorities of the users. For example, different organizations will have their own thinking of what is a usable and good schedule, and will therefore have their own set of constraints. Therefore an approach that is successful for one particular problem may not be suitable for other scenarios. As the scheduling problems in different scenarios can not be generalized due to these constraints, there is a need for diversity in the available approaches in research.

In the perspective of computer science, scheduling is generally modeled as a Combinatorial Optimization Problem. The objective in such problems, in the context of scheduling, is to find an assignment of timeslots for each of the events, so that the solution is optimal according to the given criteria. It means the problem is to find the best possible solution out of all possible solutions. So a brute-force approach which visits all possible solutions is best suited for scheduling problems; however, such approach is only applicable in a very small problem instance. As the problem size gets bigger, the search space grows exponentially, thus making this approach in-feasible. Therefore, other approaches must be considered, which can provide a solution in polynomial time. The resulting solution may not be the best possible, but may be usable for practical purpose.

The above discussion implies that new ideas and approaches for solving scheduling problems provide more opportunities towards better solutions. Keeping in view this motivation, the approach presented in this thesis is an attempt towards solving scheduling problems using the concept of evolution. This approach provides a general framework and may be used for solution of a number of scheduling problems. For the course of this thesis, the university course scheduling problem is considered.

1.2 Areas of Application

The areas of application include the following:

- Job and process scheduling in industries and production facilities,
- Course and examination scheduling in educational institutes,
- Vehicle route and tour scheduling in transportation systems,
- Frequency allocation in wireless and mobile networks,
- Rostering in military establishments and hospitals,
- Employee scheduling,
- Sporting events and tournament scheduling.

1.3 Introduction to Event Scheduling

A generalized definition of event scheduling derived from the literature is: ‘the allocation of resources to events in timeslots, in such a way that given constraints are not violated’.

Different words have been used to describe scheduling problems in daily life scenarios. A timetable describes when a particular event has to take place, for instance i) in transportation system, a timetable is a statement of when journeys are taken by vehicles on different routes. ii) In schools where a sole teacher carries out all the events of a class and where these events take place in the same room a timetable represents a sequence in which these events take place. Iii) Whereas in a university, a timetable shows the sequences of events keeping in view the availability of teachers and resources. iv) An examination schedule assigns locations on the basis of sizes of classes and facilities needed to undertake the examination. v) In

hospitals and military establishments, rosters define the assignment of duties to personnel's in a specific pattern.

All the activities related to development of such timetables, schedules or sequences are examples of event scheduling.

Timetable, schedule, roster and sequence are often considered synonymous, but literature makes certain distinctions among these terms. A **timetable** generally shows when (in term of time) particular events are to take place. A **sequence** is an order in which activities take place. A **roster** is assignment of resources to timeslots in a rotating pattern. A **schedule** includes all the specific information necessary for an activity to take place. This specific information includes:

- times at which activities are to take place,
- the order in which they take place,
- the assignment of required resources,
- any special needs of individuals or resources.

The above-mentioned terms can be formally defined as follows:

1.3.1 Timetabling

Timetabling is the allocation, subject to constraints, of given resources to objects being placed in space-time, so that a set of desirable objectives is satisfied as nearly as possible. Examples of timetabling are school class and examination timetabling and some forms of resource-to-personnel allocation.

1.3.2 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. Examples of sequencing are simple job-shop scheduling. For example, order of jobs being carried out in a factory is a sequence if jobs go through each machine in the same order.

1.3.3 Rostering

Rostering is the placing, subject to constraints, of resources into slots in a pattern. One may seek to minimize some objective, or simply to obtain a feasible allocation. Often the resources will rotate through a roster.

1.3.4 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. Common examples of scheduling are university course and examination scheduling in which many courses, labs, exams or tutorials are to be assigned, keeping in view the availability of students, teachers and other resources. Another example is transport scheduling or vehicle routing which seeks to minimize the number of vehicles or drivers. Another example is job shop scheduling which may seek to minimize the number of time periods used, and the physical resources.

Some of the above-mentioned problems may fit to more than one of the definitions, resulting in these terms often being used loosely; however, all these problems lie in the broad scope of a schedule. Therefore, we can generalize a schedule to constitute the characteristics of timetable, sequence, roster and that of any other constraints that may be involved.

1.4 Constraints involved in Event Scheduling

As discussed above, the goal of scheduling is to solve problems relating to the allocation of resources to objects being placed in space-time. The problems often involve the satisfaction of certain pre-defined conditions or objectives. These conditions are called **constraints** in terms of scheduling and are generally classified in two broad categories [1].

1.4.1 Hard constraints

These constraints must not be violated as a violation makes the schedule useless. e.g., allocation of two events needing the same resources in one timeslot is an example of hard constraint violation. Hard constraints are described as follows:

- No resource can be demanded for more than one place at any single time.

- For each time period there should be sufficient resources available for all the events that have been scheduled for that time period.

1.4.2 Soft constraints

These conditions should be satisfied if possible as violations decrease the quality of schedule. e.g., putting too much burden on one resource by using it in adjacent timeslots is an example of soft constraint violation. In real-world situations it is usually impossible to satisfy all soft constraints. These constraints represent those conditions that are desirable but not absolutely essential for a schedule. A number of soft constraints may be encountered in different scheduling problems. These soft constraints are generally categorized as under:

- **Time assignment:** An event may need to be scheduled in a particular time period.
- **Time constraints between events:** One event may need to be scheduled before or after the other.
- **Spreading events out in time:** Events may need to be scheduled in periods not consecutive to each other or on different days.
- **Coherence:** People may prefer to have the events related to them, scheduled in a number of working days and to have a number of free days.
- **Resource assignment:** Events may need some resource at a particular time or an event may need to be placed at a particular place which contains a particular resource.
- **Continuity:** Any constraints whose main purpose is to ensure that certain features of timetables are constant or predictable. For example, related events should be scheduled in the same room, or at the same time of day in every week."

1.4.3 Classification of Event Scheduling Constraints

Keeping in view the wide variety of constraints that may be encountered in different scheduling problems, scheduling constraints (both hard and soft) have been categorized into five classes [2].

1.4.3.1 Unary Constraints

These are constraints that involve just one event, such as the constraint "event *a*, must not take place on a Tuesday", or the constraint "event *a*, must occur in timeslot *b*". Both of these examples can be hard or soft constraints.

1.4.3.2 Binary Constraints

These sorts of constraints concern pairs of events, such as the event-clash constraint which states that two events can not be scheduled at one time even in different places, or those that involve the ordering of events such as the constraint "event *a* must take place before event *b*". These constraints can also be hard or soft.

1.4.3.3 Capacity Constraints

These are constraints that are governed by room capacities, etc. For example "All events should be assigned to a room which has a sufficient capacity". This particular example is usually a hard constraint.

1.4.3.4 Event Spread Constraints

These are the constraints that concern requirements such as the "spreading-out" or "clumping-together" of events within the timetable in order to ease workload, and/or to agree with timetabling policy of an organization. This is usually a soft constraint.

1.4.3.5 Agent Constraints

These are the constraints that are imposed in order to promote the preferences of the people who will use the timetables, such as the constraint "lecturer x likes to teach event a", or "lecturer y likes to have n free mornings per week". This can be a hard or soft constraint.

There is an infinitely wide range of constraints that may be encountered in scheduling problems. From practical point of view, this is understandable as different scheduling situations are likely to have their own specific needs and policies. For example, an airline may specify that more and more flights to one destination be scheduled during a specific period in order to cater for an occasion; at other time, however, it may prefer the flights to be as few as possible. For example, PIA needs to schedules more flights to Saudi Arabia for Hajj.

From a research point of view, this nature of scheduling makes it very difficult to formulate useful generalizations about the problem. The above-mentioned difficulties make the scheduling problems NP-complete in almost all variants.

1.5 NP-Complete nature of Event Scheduling Problems

Event scheduling has been known to belong to the class of problems called NP-complete, i.e., no method of solving it in a reasonable (polynomial) amount of time is yet known [1]. In [3], Cooper and Kingston have shown a number of proofs to demonstrate the NP-complete nature of scheduling problems. They provided polynomial bounded transformations from well-known NP-complete problems such as graph coloring, bin-packing, and three-dimensional matching to many different variants of the scheduling problem. In [4], Itai, Even and Shamir have also shown a transformation of the NP-complete 3-SAT problem into a scheduling problem.

The NP-complete nature means that if we want to obtain a workable schedule in a reasonable time, this will depend very much on the nature of the problem instance being tackled. Some universities, for example, may have scheduling requirements that are fairly loose: for example, there is a large number of rooms, or only a very small number of events that need to be scheduled. In these cases, maybe there is a lot of good schedules within the total search space, of which one or more can be found quite easily. On the other hand, requirements of some university might be much more demanding, and perhaps only a very small proportion of the search space will be occupied by workable schedules. In practice, the combination of constraints may often result in problems that are impossible to solve unless some of the constraints are relaxed. In case of these harder scheduling problems, more powerful methods are needed for obtaining reasonable solutions.

Scheduling problems are often compared with the Graph-coloring problem which is a benchmark NP-complete problem. A generalization of scheduling as a graph-coloring problem proves the NP-complete nature of scheduling problems.

1.6 Graph Coloring Model for Event Scheduling

A general reduction of a simple scheduling problem is given as follows:

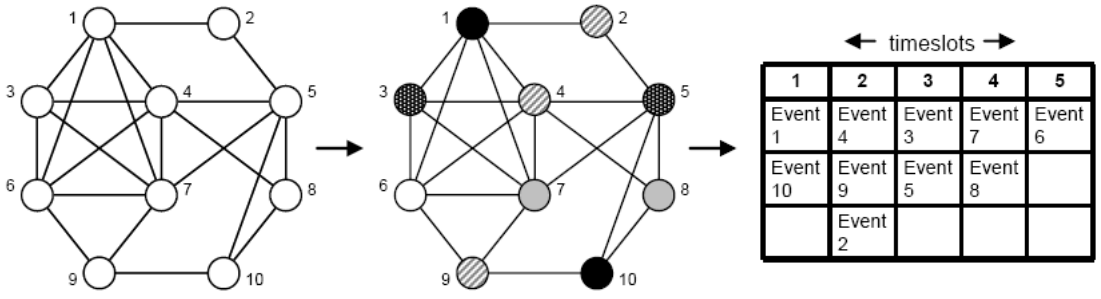
Given a simple and undirected graph G comprising a set of vertices V and a set of edges E which join various pairs of different vertices; the graph coloring problem involves finding an assignment of “colors” for each vertex in V such that

(a) no pair of vertices with a common edge are assigned the same color,

(b) the number of colors being used is minimal.

A scheduling problem can be converted into this graph coloring problem by considering each event as a vertex, and then adding edges between any pair of vertices that correspond to pairs of events that can not be assigned to the same timeslot. Each timeslot that is available in the scheduling problem then corresponds to a color, and the problem is to find a solution that uses no more colors than number of available timeslots.

In graph coloring, the term “chromatic number” is used to refer to the minimum number of colors that are needed to feasibly color a problem instance. In simple scheduling problems this also represents the minimum number of timeslots that are needed for possibly a clash-free schedule. Identifying chromatic number is an NP-hard problem. The NP-complete version of this problem defines a similar task, but in the form of a decision problem, i.e., given G = (V, E) and a positive integer k < n; is it possible to assign a color to each vertex in V such that no pair of adjacent vertices has the same color, and by only using k colors?



Given a scheduling problem, it is first converted to graph coloring equivalent. In this example, there are 10 events (10 vertices are to be colored).

A solution is then found for this graph coloring problem.

The solution is then converted back to a schedule, such that each color represents a timeslot.

Figure 1.1: Graph coloring model for solving a simple scheduling problem (with event-clash constraint only)

Another similarity between these problems involves the identification of features known as cliques (A clique is a collection of vertices that are mutually adjacent, such as vertices 1, 3, 4, 6, and 7 in fig. 1.1, which is a clique of size 5.). Graph coloring problems that reflect real-world scheduling instances will often contain large cliques. This is because in many problems there may be a large collection of events that must not be scheduled together. In the equivalent graph coloring problem, the vertices that represent these events will form a clique, and no two vertices in this clique may be assigned the same color (all of the corresponding events will need to be assigned to different timeslots). Thus, if we are given a graph coloring instance that has a maximum clique size of C, then at least C colors will be needed to color the graph legally. The task of identifying the maximally-sized clique is also an NP-hard problem [5]. This conversion to pure graph coloring problems only exists when we are considering constraints regarding conflicting events such as the event-clash constraint. When other sorts of constraints are also being considered then this will add extra complications. Regardless of this, all scheduling problems still include graph coloring problem in some form or another.

1.7 Academic Scheduling

Academic scheduling involves the task of assigning a number of events, such as classes, exams, labs, tutorials etc., to a limited set of timeslots and rooms, in accordance with a set of constraints. There are three main classes of academic timetables [6].

1.7.1 School Scheduling

The weekly scheduling for all the classes of a school, avoiding a teacher taking two classes in the same time, and vice versa. This schedule is a timetable describing when each class is taught a particular subject/lesson and in which room it is held. Each class is a set of students and generally a specific teacher is assigned to carry out all the activities of the class. Teachers are allocated the classes before scheduling. Classes are assigned specific rooms. The problem is to arrange the meetings of teachers with classes in particular time periods. Each class or teacher may be engaged in one subject at a time. Some soft constraints may be involved. This is the simplest form of academic scheduling.

1.7.2 Examination Scheduling

It is the scheduling for the exams of a set of courses, avoiding overlapping exams of different courses having common students, and spreading the exams for the students as much as possible. An exam schedule defines when each class undertakes examination of all its subjects. There is only one exam for each subject and the subjects having common students must be scheduled in different timeslots. All examinations are to take place in limited number of rooms and during a limited time period. There is a limit on number of examinations that a student should be asked to take in a single day. More than one class can take exam of different subjects in one room.

The problem requires assignment of exams to rooms and timeslots within a given amount of total time, keeping in view student clashes. In universities where there is a large number of courses, and students are allowed to take courses from a number of electives, the task of exam scheduling becomes very complex.

1.7.3 Course Scheduling

It is the weekly scheduling for all the lectures of a set of university courses, minimizing the overlaps of lectures of courses having common students. This schedule describes when and where each course is taught, keeping in view student-clashes and all other involved constraints. The problem consists of scheduling a set of lectures for each course, within a given number of rooms and in given time periods.

The main difference with the school problem is that university courses can have common students, whereas school classes are disjoint sets of students. If two classes have common students then they conflict, and they cannot or should not be scheduled at the same period. In addition, in the university problem, availability of rooms, their size and equipment play important roles, whereas in the high school problem each class has its own room.

Course scheduling differs from exam scheduling as in latter, multiple events can be scheduled in the same room at the same time provided seating-capacity allows, while in the former case, only one event is allowed in a room at one time. A

second common difference between the two concerns the timeslots. The course timetabling problems generally involve assigning events to a fixed set of timeslots, while exam timetabling problems might sometimes allow some flexibility in the number of timeslots being used.

1.7.4 Constraints involved in Course Scheduling

A number of hard and soft constraints are involved, especially in the exam and course scheduling problems. As discussed earlier, hard constraints have a higher priority than soft, and must be satisfied for the schedule to be usable. Soft constraints normally define the quality of a schedule as per the policies of the institution.

The most common hard constraint in academic scheduling is the “event-clash” constraint. This constraint states that a person is required to be present in a pair of events, then these events conflict, and therefore, must not be assigned to the same timeslot as such assignment will result in this person having to be in two places at one time. This particular constraint can be found in all the university scheduling problems. However, a great number of other constraints, both hard and soft, are involved in academic scheduling. Different universities and institutions have their own sets of specific constraints, keeping in view their policies and routines. Because of this diversity in constraint set, different instances of scheduling problems have varying level difficulty to solve.

1.8 Problem Formulation

In this thesis, the focus is on an academic scheduling scenario, generally known as the university course scheduling problem. The version of this problem being considered was originally defined in 2001 by the “Meta-heuristics Network” [7]. In 2002, it was also used for an International Timetabling Competition [8]. The problem has since become a benchmark for research in the field of course scheduling. It is a simplified form of typical real-world timetabling problems, including their common aspects. A formal description of the problem is given in the next section.

1.8.1 Problem Specification

A problem instance consists of a set *E* of *n* events that are to be scheduled into a set of timeslots *T* and a set of *m* rooms *R*, each with an associated seating capacity. A set of students *S* is also given, and each student in *S* is required to attend some subset of *E*. Events are said to conflict with each other when a student is required to attend them simultaneously. Finally, a set of room features *F* is given, which are intended to represent real-world features such as writing board, computing facilities, audio-visual facility etc. Certain features are required by each event and are satisfied by certain rooms.

A set of constraints applies to the problem, consisting each of three hard and soft constraints. The hard constraints are described as follows:

HC1: No student is required to attend more than one event at any one time. i.e., conflicting events should not be assigned to the same timeslot;

HC2: All events are to be assigned to suitable rooms. i.e., all of the features required by an event are satisfied by its very room, which must also have an adequate seating capacity;

HC3: Only one event is assigned to any one room in any timeslot. i.e. no double-booking of rooms is allowed.

In addition to the hard constraints listed above, there are also three soft constraints to be considered. These are as follows:

SC1: No student should be required to attend an event in last timeslot of day;

SC2: No student should attend more than two events in a row;

SC3: No student should have a single event in a day.

In order for a schedule to be *feasible*, it is necessary that every event *e1,…,en* is assigned to exactly one room *r1,…,rm* and exactly one of *t* timeslots (where in all cases $t \leq 45$, which is to be interpreted as five days of nine timeslots), such that the three hard constraints are satisfied. A solution is *perfect* if (a) it is feasible and (b) it has no violations of the three soft constraints.

The notions used in problem specification are summarized in the following table.

Name	Description
n	Number of events in the problem instance.
m	Number of rooms in the problem instance.
t	Maximum number of timeslots in a feasible solution (in all cases, *t* is a constant 45, comprising five days of nine timeslots).
Feasible schedule	A schedule in which all courses are assigned to timeslots and rooms and no hard constraint is violated.
Perfect schedule	A schedule which is *feasible* and in which no soft constraint is violated.

Table 1.1: Notions used in Problem Specification

1.8.2 Specification of Goals

Following goals and conditions have been defined for the formulated problem:

Goal 1: Allocation

To develop a feasible solution.

Goal 2: Optimization

Take this feasible solution as close to the perfect solution as possible.

Conditions:

- These goals must be achieved in a pre-defined time limit.
- A feasible solution with more soft constraint violations has a higher priority than an in-feasible solution with less soft constraint violations.

1.8.3 Solution Evaluation

A solution is judged by the number of constraint violations it contains. In the case of hard constraints, the term distance-to-feasibility is used to evaluate the solution. Different functions are used to calculate distance-to-feasibility, keeping in view the problem instances in use, most common being the number of courses remaining un-allocated. For soft constraints, the evaluation is done by calculating the

total number of penalty points in the solution. Penalty points are calculated in the following way:

- For SC1, if a student has a class in the last timeslot of the day, it is counted as one penalty point. i.e., if there are x students in this class, x number of penalty points are counted.)
- For SC2, if one student has three events in a row, it is considered as one penalty point. If a student has four events in a row, two points are counted, and so on.
- For SC3, each time a student is encountered with a single event on a day, one penalty point is counted.

Chapter 2: Literature Survey

Due to its significance, scheduling has become an application area with rich knowledge and experience. A number of studies have been carried out and many algorithms have evolved for scheduling problems. In this chapter, a survey of different techniques is presented. As the approach presented in this thesis lies in the meta-heuristic framework, main part of this chapter is devoted to meta-heuristic approaches and their classifications in literature.

2.1 Overview of Research in Academic Scheduling

Many early techniques used in scheduling algorithms were directly derived from graph-based heuristics, because of their obvious similarities [9]. An early example of such algorithm was provided in [10]. This approach was used for several years at the University of Ottawa in the 1970's, and it is said to be capable of scheduling 390 events involving 16,000 students into 25 timeslots. Another early example is the EXAMINE timetabling system documented in [11]. In this paper, the system is applied to a set of real-world exam timetabling problems taken from a number of different universities. These problem instances are now referred to as the Carter Instances and have been used in many exam timetabling papers. Another heuristic based approach that models scheduling problem on graph-coloring theme is given in [12].

Other early approaches to scheduling problems have involved using constraint-based techniques [13] and also integer programming [14]. In the near past, research in scheduling problems has been mainly focused on meta-heuristic based techniques in which intuitive problem-specific heuristics have been used to reduce the number of solutions processed. Some of these techniques considered in this thesis are Fuzzy Heuristic Ordering by Asmuni and Burke in [15], Graph-Based Hyper Heuristic by Rong Qu and Burke in [16], Variable Neighborhood Search by Abdullah and Burke in [17], Tabu-search Hyper Heuristic by Kendall, Soubeiga and Burke in [18], Local search by Socha in [19], Ant Algorithms by Socha in [20] and Genetic Grouping by Ben Paetcher in [21].

2.2 Approaches to Automated Scheduling

2.2.1 Sequential Methods

These methods order events using problem-specific heuristics and then assign the events sequentially into timeslots so that no events in the period are in conflict with

each other [Car86]. In sequential methods, timetabling problems are usually represented as graphs, and construction of a conflict-free timetable can be modeled as a graph coloring problem. Some of the earliest works towards automated scheduling have used this approach. As indicated above, algorithm in [10] operates by using largest-first type heuristic to select the events for assignments, keeping in view the event-clash constraint and new timeslots are opened when needed. Different heuristics are used to minimize the soft constraints. The EXAMINE timetabling system [11] is also based on a backtracking sequential-assignment. A number of variants are tested by the authors, and best performance is usually gained when two procedures are followed: firstly, when those events are inserted into the timetable first which have the highest number of colors adjacent to them; and secondly, when an additional algorithm is also used to identify large cliques in the problem, so that the events within these cliques can then be given priority. The backtracking feature of algorithm enables it to undo previous assignments of events to timeslots when no feasible timeslot there remains for an un-assigned event.

2.2.2 Constraint Based Methods

In these methods a scheduling problem is modeled as a set of variables (events) to which values (resources such as rooms and timeslots) have to be assigned to satisfy a number of constraints [6, 13, 22]. Usually many rules are defined for assigning resources to events. When no rule is applicable to the current partial solution, a backtracking is performed until a solution is found that satisfies all constraints. Another constraint based technique [23] models scheduling problems as Constraint Satisfaction Problems because of large number of complex constraints involved.

2.2.3 Knowledge Based Methods

The objective of using knowledge based techniques is to model the human knowledge for solution of computational problems. An early approach using knowledge based techniques and constraint networks on real-world employee scheduling was presented in [24]. The problems were explicitly represented on some constraints in the constraint based processing and rules were incorporated into the scheduling process. The preliminary results showed that the explicit representation and the ordering heuristic are efficient for solving employee timetabling problems. In

[25], the authors designed a timetable scheduler that used the knowledge modeled as rules, incorporated with heuristics, within course scheduling process to schedule data that was stored in separate bases. The results so obtained were promising for real world scheduling problems and authors claimed that the scheduler was flexible and general and was applicable to other course scheduling problems with the use of an object-oriented methodology. In [26], the authors proposed a conceptual model within which knowledge was modeled into heuristics that applied the rules to guide scheduling process for course scheduling problems. Recent knowledge based techniques have used expert system to model knowledge of scheduling as rules.

2.2.4 Local Search Methods

A large number of studies have used local search for scheduling problems. The term local search or neighborhood search expresses the idea that these algorithms modify an inconsistent assignment to move to a better assignment. During iterations, only assignments from the neighborhood of the current assignment are considered and one of them is picked [27]. In general, local search algorithms are incomplete and do not guarantee of finding a complete assignment satisfying all the constraints. Therefore, they have generally been used in conjunction within hybrid frameworks. These algorithms may be more efficient with respect to response time as they are guided by heuristics. Local search approach adopted by [19] has showed impressive results.

There are many ways to define neighborhood of an assignment [28]. Two basic local search algorithm schemes, hill-climbing and min-conflict, usually start from a randomly or heuristically selected assignment which repeatedly performs local steps to their neighborhood till a solution is found or the time limit exceeds. But, they differ in the manner how the neighbor assignments are selected. Hill-climbing always selects a better assignment out of all the neighbors (the assignment which minimizes the number of violated constraints). When an assignment better than the current one is not available, the search is stuck in a local optimum and the algorithm usually restarts from another initial randomly selected assignment. A variation steepest-ascent hill climbing, selects the best possible assignment out of all the neighbors.

On the other hand, min-conflict algorithm chooses the best assignment only from a subset of the neighbor assignments. Usually, it randomly selects any variable that is involved in an unsatisfied constraint, and then picks a value which minimizes the number of violated constraints. If no such value exists, it randomly picks a value that does not increase the number of violated constraints. If algorithm reaches a strict local minimum, it does not perform any move at all and it does not get terminated. To deal with this problem, a variation, Min-conflict Random Walk Algorithm is used in literature.

2.2.5 Cluster Methods

In these methods the set of events is split into groups which satisfy hard constraints and then the groups are assigned to time periods to fulfill the soft constraints. Different optimization techniques have been employed to solve the problem of assigning the groups of events into time periods. The main drawback of these approaches is that the clusters of events are formed and fixed at the beginning of the algorithm and that may result in a poor quality timetable.

2.2.6 Hyper-Heuristic Methods

Hyper-heuristics are “heuristics that choose heuristics” [29]. The main difference between hyper-heuristics and the widely used meta-heuristics in scheduling is that hyper-heuristics is a method of selecting heuristics from a variety of different heuristics that may include meta-heuristics. So hyper-heuristics are more general purpose methods. In [21], the authors have obtained good results by using genetic algorithms to select from a set of heuristics encoded in the search space. An approach was presented in [2], on open-shop scheduling problems using genetic algorithms to search a space of abstractions of solutions to “evolve the heuristic choice”. In a real-world scheduling problem, genetic algorithms are used to construct a schedule builder that chooses the optimal combinations of heuristics [30]. Another approach in [31] has used a genetic algorithm selecting the heuristic to order the exam in a sequential approach for exam timetabling. Another approach using fuzzy logic to apply an ordering of heuristics has been developed by Asmuni and Burke in [15].

2.2.7 Decomposition Methods

Real-world scheduling problems are generally very large and complex. To address this problem, decomposition and partition techniques have also been studied with some success. The basic idea is to decompose the problem into a set of sub-problems that are small enough to be solved by using simple approaches. Then these sub-solutions are combined for the original problems. In [32], an algorithm has been presented which decomposes the course scheduling problems into a series of easier assignment-type sub-problems. An approach of decomposing the timetabling data to produce shorter flexible length timetables was also studied in [33]. In [34], the authors have employed a multi-stage algorithm in an evolutionary approach using graph coloring heuristics to solve examination timetabling problems that were decomposed, and while the sub-problems were solved by using a memetic approach.

2.3 Meta-Heuristic Methods

According to the Meta-heuristics Network, a meta-heuristic is a general framework which may be applied to different optimization problems with some modifications needed according to specific problem scenarios. Due to this generalized nature, these techniques have become increasingly popular in trying to solve scheduling problems. In the recent past, a number of meta-heuristic techniques such as simulated annealing, tabu search, iterated local search, evolutionary and genetic algorithms, ant colony optimization and other hybrid approaches have been investigated for scheduling. These techniques begin with one or more initial solutions and employ search strategies that try to avoid local optima. These algorithms have produced good solutions on different problem instances.

2.3.1 Tabu Search

Tabu search uses local search along with a mechanism to avoid getting trapped in a local minimum. The mechanism is based on a tabu list, which is a special short term memory containing pairs of variable and values, which is used to maintain a history of previously encountered assignments. The assignments in the tabu list are not considered for the next iterations. This mechanism prevents the search from being trapped in local optima.

In course scheduling, tabu Search was mainly investigated on real-world problems in different institutions with specific requirements. Good results were reported with different variations of tabu list, initial solutions and objective functions, etc [35]. In [36], the authors have developed a tabu search based general problem solver for a range of constraint satisfaction problems including a high school timetabling problem. Results achieved were competitive as compared with others. Approaches that integrated tabu search with other techniques were also investigated. In [18], a tabu search hyperheuristic has been developed for timetabling and rostering. The results obtained were better than using either method alone. Research on examination timetabling problems was carried out in [37], which studied different aspects (length of tabu lists, representations and initialisation methods of solutions) of utilising tabu search.

2.3.2 Simulated Annealing

This method simulates the physical process of annealing. In annealing, a material is heated and then cooled, usually for softening and making the material less fragile. Simulated annealing exposes a solution to ‘heat’ and then to ‘cool’ it for producing more optimal solution, i.e. an in-feasible solution is taken and random variations are applied to achieve good solution. A worse variation is accepted as the new solution with a probability that decreases as the computation proceeds. The search tries to avoid local minima by jumping out of them early in the computation. Toward the end of the computation, when the temperature or probability of accepting a worse solution is nearly zero, this simply seeks the bottom of the local minimum. The chance of getting a good solution can be traded off with computation time by slowing down the cooling schedule. The slower the cooling, the higher is the chance of finding the optimum solution, but the longer the run time. Thus effective use of this technique depends on finding a cooling schedule that gets good enough solutions without taking too much time. Literature suggests that the implementation is highly dependent on various settings and parameters (e.g. solution space, cooling schedule, neighborhood generation, cost function) on both examination and course/school scheduling problems thus careful selection of parameters and settings on this algorithm are needed.

2.3.3 Genetic/Evolutionary Algorithms

An evolutionary algorithm is a generic population-based meta-heuristic optimization algorithm. An evolutionary algorithm uses some mechanisms inspired by biological evolution: reproduction, mutation, recombination, and selection. Candidate solutions to the optimization problem play the role of individuals in a population, and the fitness function determines the environment within which the solutions live. Evolution of the population then takes place after the repeated application of the above operators. Usually, an initial population of randomly generated candidate solutions comprises the first generation. The fitness function is applied to the candidate solutions and to any subsequent offspring. In selection, parents for the next generation are chosen with a bias towards higher fitness. The parents reproduce by copying with recombination and/or mutation. Recombination acts on the two selected parents (candidates) and results in one or two children (new candidates). Mutation acts on one candidate and results in a new candidate. These operators create the offspring (a set of new candidates). These new candidates compete with old candidates for their place in the next generation (survival of the fittest). This process can be repeated until a candidate with sufficient quality is found or a previously defined computational limit is reached.

Genetic and evolutionary algorithms have been widely studied by researchers in scheduling, concerning different aspects of timetabling problems. In course scheduling, [38] investigated a parallel genetic algorithms that greatly reduced the execution time to solve the problem. Approaches that hybridise genetic algorithms with local search techniques during the evolution, which are known as Memetic Algorithms have been investigated and promising results have been obtained [39]. Initialization is one of the important issues in genetic algorithms and evolutionary algorithms. In [21], Ben Paechter has used different methods like group-based operators to achieve good result on some instances.

2.3.4 Ant Algorithms

In these algorithms, artificial ants try to solve a problem by adopting the behavior of real ants [40]. The inspiration behind ant algorithms is the ability of ant groups or colonies to perform well coordinated activities. Foraging behavior and capability of ants to find shortest path between food and home has been adopted for

solution of many problems of combinational and computational nature. [41] Describes that ants communicate to each other with the help of a scent called pheromone. When ants search for food, they form a path by leaving behind the pheromone. An ant going on the shortest path between food and home, will deposit more pheromone than an ant going on longer path, as it moves to and fro, on the path more for number of times. Other ants, when encounter a more pheromone-rich path than their current path, divert to this path. Thus, all ants are directed to the shortest path. Ant algorithms work on a common memory space which is taken as pheromone trail. All ants update their knowledge on this memory. Good solution receives more pheromone and bad solutions are eliminated by pheromone evaporation. This shared memory allows the ants to find solutions rapidly. Pheromone evaporation provides a way to escape deadlocks. Soch, Knowles and Samples [42] have used max-min ant system to achieve best known results on benchmark instances of the course timetabling problem by simulating a colony of ants, wherein each ant constructs a complete candidate timetable by placing courses one-by-one in a predefined order. The selection of timeslot to assign is done keeping in view the pheromone level. One candidate timetable is then selected on basis of a fitness function from the set of timetables generated by the ants. Pheromone levels are updated. This process is iterated till the time limit and finally the best solution is selected.

2.3.5 Classification of Meta-Heuristic Methods

In [43], R. Lewis categorizes meta-heuristic algorithms for scheduling into three categories. An overview of each category and various algorithms which have used these approaches are described as follows:

2.3.5.1 One-Stage Optimization Algorithms

Scheduling algorithms of this type allows the violation of both hard and soft constraints, with an aim to search for a solution that has a sufficient satisfaction of both. Some mechanism of weight-age has to be used to give hard constraints higher priorities than that to soft constraints. This kind of approach is generally easy to implement as any type of constraint may be easily incorporated in the problem by defining a suitable priority-weighing function. Such implementation is also easy because only a single weighing function is to be used for searching a solution.

Because of the convenience to incorporate a complex set of constraints, this approach has been used in a large number of scheduling algorithms.

2.3.5.2 Two-Stage Optimization Algorithms

The characteristics of two-stage optimization algorithms for scheduling may be summarized as follows:

In stage-one, the soft constraints are generally ignored, and only the hard constraints are considered i.e. only a feasible solution are required. In the next stage, assuming feasibility has been found; attempts are made to minimize the number of the soft constraint violations. In this step, violation of hard constraints is not allowed. An immediate benefit of this technique is that there is no need to define a weight-age function. Such approach seems more reasonable, as achieving a feasible schedule is the main aim, and no compromise is made on hard constraint violation when reducing soft constraints. Majority of the above-mentioned algorithms fall in the class of two-stage algorithms.

2.3.5.3 Algorithms that allow Relaxations

In these methods, some aspect of the problem is relaxed so that the soft constraints may be satisfied but not at the expense of violating any of the hard constraints. Two common ways to provide relaxation are:

- Events that cannot be feasibly assigned to any place in the current schedule are left unplaced. The algorithm then attempts to satisfy the soft constraints and tries to assign these unplaced events to somewhere in the schedule at a later stage.
- Extra timeslots are opened in order to deal with events that have no existing feasible timeslot available. The algorithm then tries to reduce the number of timeslots down to the required amount, while taking into consideration the satisfaction of the soft constraints.

2.4 Comparison of Scheduling Approaches

Comparisons concerning a range of issues in heuristic and meta-heuristic methods for timetabling have been carried out. In [44], Ross and Corne compared genetic algorithms, simulated annealing and stochastic hill climbing on a collection of

real scheduling problems, concerning the solution quality and number of useful solutions. The conclusions were that the stochastic algorithms perform generally well with respect to the solution quality. Some other comparisons among simulated-annealing, tabu search, genetic algorithms and memetic algorithm (genetic algorithms with local search) have suggested that tabu search generally obtains the best result, and genetic algorithm with local search is capable of giving a set of good quality solutions thus is much flexible to users who may have a variety of objectives. However, different algorithms within specific circumstances may perform differently on particular scheduling problems. In general, genetic/evolutionary algorithms are able to give a number of useful distinct solutions thus in real-world problem solving, they may be more flexible on providing the users solutions that satisfy different aspects of requirements.

A comprehensive comparison of different meta-heuristics for course scheduling has been presented by Rossi-Doria et al. [45] who provided a comparison between proposed five different meta-heuristic based algorithms (evolutionary algorithms, ant colony optimization, iterated local search, simulated annealing, and tabu search). Some of these algorithms attempted satisfaction of both hard and soft constraints simultaneously. Other algorithms, such as the iterated local search and simulated annealing approaches employed two separate steps for hard and soft constraints. In this comparison, it was observed that in the cases where feasibility was generally achieved, the algorithms using two steps tended to produce better results. Keeping in view their observations, the authors offered two conclusions:

- “The performance of meta-heuristic, with respect to satisfying hard constraints and soft constraints, may be different;
- “A hybrid algorithm consisting of at least two phases, one for taking care of feasibility, and the other for taking care of minimizing the number of soft constraint violations, is a promising direction.”

2.5 Summary

A number of studies has been carried out and many algorithms have evolved for scheduling problems; both problem-specific and generalized. Problem-specific approaches, such as graph theory and integer programming, can produce reasonable

solutions for smaller scheduling problems. However, they are generally not capable of dealing with problems with larger size and complex constraints. More generalized techniques such as meta-heuristics (simulated annealing, evolutionary algorithms, and tabu search etc.) have been reported to obtain better results on a wide range of problems of different sizes. Problem- specific heuristics must be used in their support to reduce the number of possible solutions processed and to fit them into specific problem scenarios. Some important observations are as follows:

- No particular approach is superior to any other on all occasions. It is likely that certain approaches might be more suited to certain types of problems and certain types of user requirements.
- The performance of a meta-heuristic, with respect to satisfying hard constraints and soft constraints may be different. Therefore, hybrid algorithms consisting of separate stages and techniques for hard and soft constraint satisfaction provide better chance of a good solution.
- There is a better chance of achieving a good solution, when events are inserted into timeslot in some order, e.g. which have the highest number of common resources.
- A mechanism to provide relaxation is helpful to avoid dead-end.

Chapter 3: Die-hard Cooperative Ant Behavior Approach

3.1 Overview and Inspiration

As observed in the literature survey, an approach comprising of two stages; first allocation and then optimization, performs better on course scheduling problems. Keeping in view this observation, this two-stage framework is devised, based on a typical ant behavior. Different techniques are used within this framework. The resulting algorithm is named “Die-hard Cooperative Ant Behavior Approach” (DCABA). This approach is different from the ant approaches previously used for solving scheduling problems. The main inspiration of this approach is the following ant behavior:

- Ants explore the surroundings of their colony in search of food by roaming about randomly. When ants find food, they try to take it home. If one ant can pick up the piece of food, it takes it alone. If food is heavy, more ants try to pick it. If food is too heavy, ants break the piece of food and then start trying again.

In this exercise, ants show persistence and cooperation. A piece of food once approached is never left, neither for other ants nor for a later time. This approach has been adopted for the solution of course scheduling problem. A problem once encountered is emphasized upon and it is tried that it may be overcome positively. Similarly, if a prospective place is encountered, all available mechanisms are activated to take advantage of it. The approach results in a two-stage hybrid framework in which a number of heuristics and techniques may be used. In the first stage, allocation of courses is attempted. In the second stage, the quality of obtained solution is improved by optimization.

In allocation stage, a quick but less powerful mechanism starts placing courses in time space according to some simple heuristics. If it gets stuck somewhere and is unable to allocate courses any further, a set of diverse helper functions are invoked which try to overcome the bottleneck. Once they take the solution out of trap, main function starts again. If they fail to overcome the problem, the problem scenario is changed by using an operator. In optimization stage,

optimizer function starts improving the solution obtained from first stage. As in allocation stage, a helper mechanism is present to take care of bottlenecks. If it fails, another mechanism changes the scenario on the time space table and optimizer function takes charge again. The process continues till a pre-defined time limit.

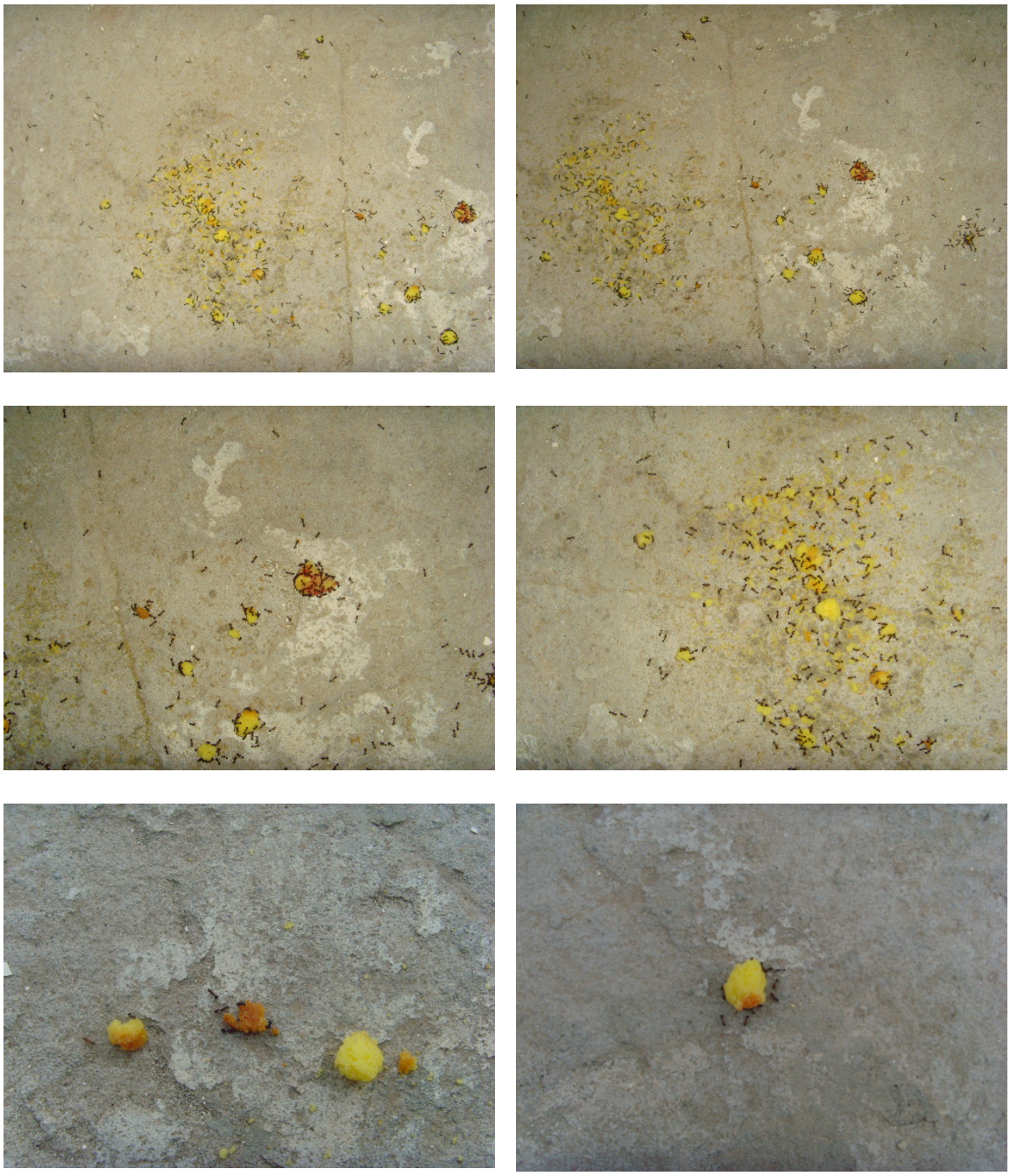


Figure 3.1: Ants carrying food

3.2 DCABA Modeling

The ant behavior to be modeled can be elaborated as follows:

a) Ants roam about randomly and find food. Shortest path to food is found by using pheromone trail.
b) Ants start taking food to home using the shortest route.
c) If an ant finds a heavy piece of food which it can not carry, it waits for help. When another ant reaches it by following the trail, both the ants try to pick the food. If both fail, they wait for more ants, and so on.
d) If a group is able to take the food home, each of these ants start working separately as before.
e) If a specific number of ants is unable to pick the food, the ants break the food into pieces and repeat the above process.

This behavior is modeled for our scenario as follows:

a) A mechanism is selected to be used as the shortest path towards solution (*SelectShortestPath*).
b) Process of seeking solution is started using the shortest path mechanism (*SolutionSeekingProcess*).
c) If some bottleneck is encountered during the *SolutionSeekingProcess*, a helping mechanism is invoked (*Helper*).
d) If the helper mechanism succeeds, the *SolutionSeekingProcess* takes over, and advances.
e) If the helper mechanism fails, orientation of the scenario is changed by invoking some suitable mechanism (*OrientationChanger*) and the *SolutionSeekingProcess* takes over. (Break of the food into pieces by ants is taken as changing the orientation of the problem).

3.2.1 DCABA Modeling for Phase-1 (Allocation)

Keeping in view the behavior modeling above, the following mechanism has been used in the allocation phase:

Step 1: Finding Shortest path for complete allocation:

A set of initial solutions is developed using a ***Heuristic-Selection Operator***. Each solution in the set is developed by using a specific heuristic. The solutions are

evaluated by an evaluator and the heuristic which gives the best initial solution is selected for use in the allocation step. Each heuristic is applied by sorting the list of courses and rooms and then starting allocation by selecting courses and rooms one-by-one from the sorted lists.

Step 2: Allocation step

An ***Allocation Operator*** starts allocating the courses in the time-space. The heuristic selected in the previous step is used to select courses and rooms for the next allocation. One period (that is one period in each day, in this case, 5 periods as number of days is 5) is opened and each course is tried to place in its feasible room in the slots opened so far. If a place is found such that placing this course here doesn't result in any clash, the course is placed, and next course is selected. If no such place is found for any of the courses in the timeslots opened so far, a set of helper functions is invoked.

Step 3: Helper Functions

The set of ***Helper Functions*** try to take the solution out of deadlock by trying to allocate the courses left from previous step. These functions basically work on local search. No violation of hard constraints is allowed at any instance of time while their execution. After running these helper functions, one more period is opened and allocation operator starts again.

Step 4a: Change orientation of problem space by random-shuffling

When helper functions fail to allocate all the courses, the orientation of the solution is changed by invoking a ***Random-Shuffle Operator*** and allocation operator starts again.

Step 4b: Change orientation of problem space by selective-shuffling

If after opening all the allowed number of periods, some courses are left un-allocated, the orientation is changed by a ***Selective-Shuffle Operator*** and allocation operator starts again. These steps are repeated till all the courses are allocated or time limit is over.

3.2.2 Functions and Operators in Phase 1

3.2.2.1 *Heuristic-Selection Operator*

This operator uses heuristics to obtain a set of initial solutions. The solutions are then evaluated to find a solution in which maximum number of courses has been allocated. The heuristic which developed this solution is selected for use. The following heuristics are used:

- Start allocation from those courses which have **least number of feasible rooms** available. This was achieved by attaching a list of feasible rooms with each course and then the list of courses was sorted, with the course having lowest number of feasible rooms coming in the first place, and the course having highest number of feasible rooms coming in the last place.
- Start allocation from those courses which **need most number of features in rooms**. A list of features needed by a course was attached with the course. The list of courses was then sorted, with the course needing most number of features coming in the first place and the course needing least number of features coming in the last place.
- Start allocation from the course which has **most number of attending students**. The list of courses was sorted by number of attending students in descending order.
- Start allocation from courses which **clash with most number of courses** (that is, it shares one or more of its attending students with those courses). List of courses was sorted such that a course which clashes with most number of courses comes first and the course which clashes with least number of other courses comes at the last place.
- Start allocation from the course which **shares most number of its attending students with some other courses**. The list of courses was sorted such that the course sharing most number of attending students with other courses comes first, and so on.
- Start allocation from the course which has the **highest ratio of clashing courses to students** (number of clashing students / number of clashing courses). The list of courses is sorted in descending order of this ratio.

- Start allocating the courses from those **rooms which are feasible for most number of total courses**. A list of feasible rooms for each course is attached with it. This list is sorted such that the room which is feasible for most number of courses comes first and so on.

3.2.2.2 *Allocation Operator*

This operator selects the next course and room for allocation from the list generated by heuristic-selection operator and places the courses in the spaces opened so far in a 3-dimensional timetable-array (periods x days x rooms). It ensures that no hard constraint is violated by checking only the empty and feasible spaces for a course under consideration. If an empty feasible place is found, the operator puts the course identifier in the 3-D array, thus indicating an allocation.

3.2.2.3 *Helper Functions*

These functions try to allocate courses by finding feasible places using local searches. Each local search is limited by a criterion. No violation of hard constraints is allowed at any instance of time during their execution. Following mechanism describes the working of this set of functions:

Function 1: Try to move an allocated course *X* to an empty feasible place *Rx* which is not accepting the un-allocated courses *C*, if it is occupying a place *Rc* which can accommodate the un-allocated course *C*.

a) Find an occupied feasible room *Rc* for *C*, where course *X* is already allocated.
b) Find an empty feasible room *Rx* for *X*.
c) If *X* can be moved to *Rx*:
    i. Move *X* to the *Rx*,
    ii. Allocate *C* to *Rc*.
d) If *X* can't be allocated to *Rx* because of a clashing course *CX*:
    i. Try to find an empty feasible room for *CX*,
    ii. If a room is found, move *CX* to that room,
    iii. Move *X* to *Rx*,
    iv. Allocate *C* to *Rc*.

Function 2: Find an empty place, and locate the allocated course *CC* which is not allowing any un-allocated courses *C* to be placed there (due to student-clash). Try to move this clashing course *CC* to somewhere else.

a) Find an empty feasible room *Rc* where *C* was not placed due to a clashing course *CC* in the same timeslot.
b) Find a feasible room for *CC*.
c) If found; move *CC* there.
d) Allocate *C* on *Rc*

3.2.2.4 *Random-Shuffle Operator*

This operator changes the orientation of solution space by shuffling courses in one the following ways:

- Each allocated course is tried to move to an empty feasible place.
- Each allocated course is tried to be swapped with another allotted course.

3.2.2.5 *Selective-Shuffle Operator*

This operator changes the orientation of solution space by moving allocated courses placed at the feasible places of any unallocated courses to some empty feasible places which can accommodate these allocated courses.

3.2.3 DCABA Modeling for Phase-2 (Optimization)

Keeping in view the behavior modeling above, the following mechanism has been used in the optimization phase:

Step 1: Finding Shortest path for optimization:

It is assumed that the solution can be reached fairly fast, if those moves are favored which decrease most number of problems as compared to other optimizing moves possible in a certain scenario. So ***Steepest-Ascent Hill Climbing*** is used as the shortest path towards solution.

Step 2: Optimization step

An ***Optimization-Operator*** starts searching the solution space for improvements using steepest-ascent hill climbing.

Step 3: Helper Mechanism

The optimization-operator in step-2 comprises of two functions. Each of these functions is run as long as it can improve the solution. If it fails to improve the

solution, second function is called and vice versa. So both functions help each other by jumbling the solution and providing new scenario to other function when they perform moves to improve solution.

Step 4: Change orientation of solution space

When the optimization-operator and helper mechanism fail to improve the solution any further, orientation of problem is changed by introducing a ***Least-Penalty-Shuffle Operator***.

3.2.4 Functions and Operators in Phase 2

3.2.4.1 *Steepest-Ascent Hill Climbing*

This is a local search approach which favors the next prospective solution which is closest to the desired/optimal solution. In the case of optimization in course scheduling problem, it means favoring a move which decreases most number of soft constraint violations from the set of all moves.

3.2.4.2 *Optimization-Operator*

This operator consists of two functions. These functions are similar to the neighborhood operators used by Abdullah, Burke and McCollum [17] in their Variable Neighborhood Search.

- **Function 1** swaps entire periods in a day with other periods in any day. These moves can not result in introducing hard constraints as the whole cluster of allocated courses in a period in a day is swapped. All swapping moves are evaluated and only that move is committed which decreases most number of problems.
- **Function 2** shuffles each allocated course with other allocated courses keeping in view the hard constraints. The shuffle move, which decreases most number of problems, is committed after evaluating all possible moves.

3.2.4.3 *Least-Penalty-Shuffle Operator*

This function changes the orientation of solution space with the help of an analyzer agent. This agent analyzes the timetable array and finds out which allocated courses are causing problem. The least-penalty-shuffle operator then moves these problem courses to a different slot by shuffling with already placed

course or an empty place. The violation of hard constraints is always checked and avoided by only looking for candidate place for shuffling in the list of feasible places of the problem course under consideration. For each course, all possible moves are evaluated and the move which introduces least number of problems is selected.

3.3 Important characteristics of DCABA

- DCABA works on a hybrid framework in which many heuristics and techniques may be used. This can give more diversity in options to solve multi-constrained problems where different techniques specific for certain constraints may be designed.
- DCABA may be customized to meet the needs of a specific university, as helper functions of different capability may be tailor-made to suit some specific constraints or to give more priority to a specific requirement (e.g. allocation capability may need to be more powerful than optimization in some scenarios).
- In optimization phase, a solution which is better than previous solutions is saved. It ensures that the best solution which is reached at some instance of time in optimization phase is available when the algorithm stops. It is helpful as the least-penalty-shuffle may result in a more problematic solution than a previously reached better solution.
- One period at a time is opened up and allocation is tried for as many courses as possible (i.e. emphasis on using least number of periods). So at the end, it is probable that one soft constraint (class scheduled in the last period of day), is satisfied.
- At the end of allocation phase, a function tries to move classes in the last period of all days to first periods. This works as the first optimization step as it tries to decrease the last period problem.

Chapter 4: DCABA Implementation

4.1 Implementation Details

A simulator based on DCABA is developed in C#. Some important characteristics of the implementation are as follows:

- A three-dimensional matrix/array [*i* x *j* x *k*] is used to represent the schedule.
    - *i* rows represent periods,
    - *j* columns represent days,
        - *i* x *j* is a timeslot.
    - *k* rooms available in a timeslot.
- Each cell represents a room in a timeslot.
- Course allocation is done by assigning a number corresponding to a course to a cell in the matrix. -1 represents an empty cell. This way, it is ensured that more than one course cannot be assigned to a cell, which means one of the hard constraints is never violated.
- At the start of execution, a data structure related to the problem instance being input is created. This data structure contains following matrices:
    - Course-student matrix indicates which students are registered for a course.
    - Course-room matrix indicates which rooms are feasible for a course.
    - Course-feature matrix indicates which features are required for a course.
    - Student-course matrix indicates which courses a student is taking.
    - Room-feature matrix indicates which features are provided by a room.
- Search space is limited with the help of these matrices, e.g. when searching for a feasible place for a course, only its feasible rooms are searched.

4.2 Input/Output

As given in the problem specification, total number of available timeslots is 45 (5 days of 9 hours each) and it is hard-coded in the implementation. Problem

instances are provided in the form of text files. Each instance contains the following information:

- Set of courses to be scheduled.
- Set of rooms in which courses can take place.
- Size of each room.
- Set of students attending the courses.
- Set of features satisfied by rooms.
- Set of features required by courses.

The solution is also output as a text file containing the allocation of courses in timeslots and rooms. Each problem instance and solution is represented by separate text files.

4.3 Experimental Setup

Three studies are performed on the simulator. First study tests the overall capability of the algorithm in both phases; allocation and optimization, while the second study tests only the allocation capability. The third study investigates the effect of using individual Heuristic Operators on the allocation capability. Study-2 and 3 use same problem instances while study-1 has its own set of problem instances. A limited time is available to the simulator to run the instances, according to the characteristics of individual instance sets. There are different methods of evaluating the solution for all the three studies.

4.3.1 Study 1: Allocation + Optimization

In this study, both phases of the algorithm are tested. The available time has to be managed to first achieve a feasible solution and then optimize it. Eleven instances are run in this study. Details about these instances are given next. Results are compared with Fuzzy Heuristic Ordering by Asmuni and Burke [15], Graph-Based Hyper Heuristic by Rong Qu and Burke [16], Variable Neighborhood Search by Abdullah and Burke [17], Tabu-search Hyper Heuristic by Kendall, Soubeiga and Burke [18], Local search by Socha [19] and Ant Algorithm by Socha [20].

4.3.1.1 Problem Instances

These Instances are designed to test both allocation and optimization capability of algorithms.

Instance Name	Courses	Rooms	Features	Students	Average Student/course	Average Course/Student	Average Room Option	Courses With One Room option
small1	100	5	5	80	7	9	1	48
small2	100	5	5	80	8	10	1	61
small3	100	5	5	80	7	9	2	52
small4	100	5	5	80	5	6	2	25
small5	100	5	5	80	9	11	2	34
medium1	400	10	5	200	8	17	4	21
medium2	400	10	5	200	8	17	3	24
medium3	400	10	5	200	8	17	3	21
medium4	400	10	5	200	8	17	3	27
medium5	400	10	5	200	8	17	2	88
large	400	10	10	400	17	17	1	288

Table 4.1: 11 Problem Instances for Study 1

4.3.1.2 Mechanism for Obtaining Results

In this study, the aim is, to first find a feasible solution and then optimize it to make perfect solution (with no soft constraint violations). So, the quality of a solution can be judged by the number of soft constraints in it. If number of soft constraints is zero, the solution is perfect. As the number goes higher the quality of solution is lower. It also means that hard constraint violations are not allowed. Thus, if a solution has any hard constraint violations or any courses are unplaced, then it is left out of the competition as even the allocation is in-complete.

A benchmarking utility [46] is also provided with the problem instances to judge the quality of solution. The solution, in the form of text file, is input to this utility

and it returns a number corresponding to the number of soft constraint violation left in the solution. An infeasible solution is not accepted by the utility. Another benchmarking utility [47] is provided with the problem instances used for determining the time for which an algorithm may run on the simulating CPU.

4.3.2 Study 2: Allocation

Only allocation capability of the algorithm is tested in this study. i.e., phase-1 of the algorithm is run only and all available time is utilized in attempt of allocation of courses. Sixty instances are run in this study. Details about these instances are given next. Results are compared with Heuristic Search Algorithm and Grouping Genetic Algorithm of Rhydian Lewis and Ben Paechter [21].

4.3.2.1 Problem Instances

These are Hard-to-Solve Instances [48] with respect to allocation and are generated by a problem generator by "Rhydian Lewis and Ben Paechter" to check allocation capability of their algorithms mentioned above. The instances are divided into three sizes:

Instance	Average Number Of Courses	Average Features Required per Course	Average Number of Students per Course	Average Number of Clashing Students	Avg. Number of Courses with only One Room Option
Small	211.5	0.9	79.15	728.94	62.1
Medium	403.25	1.9	37.1	572.1	177.8
Large	1023.75	4.15	21	565.45	617.3

Table 4.2: Characteristics of Instances for Study 2

Following observations are made about these instances:

- Small Instances have fewer courses, low feature requirements and large number of clashing students.
- Medium instances have fewer courses, medium number of clashing students and less feasible rooms.
- Large Instances have large number of courses, more features needed and less feasible rooms.

Instance Name	Courses	Rooms	Features	Students	Average Student/course	Average Course/Student	Average Room Option	Courses With One Room option
Big1	1000	28	20	1000	15	15	1	763
Big2	1000	25	20	1000	17	17	1	803
Big3	1000	25	20	900	15	17	1	710
Big4	1050	25	20	800	15	20	2	648
Big5	1075	25	20	1000	18	20	1	818
Big6	1075	25	20	1000	20	22	1	841
Big7	1050	25	20	1100	25	24	1	948
Big8	1025	25	20	1000	19	20	1	608
Big9	1050	25	20	800	15	20	2	678
Big10	1075	25	20	1000	18	20	1	673
Big11	1075	25	20	1000	18	20	1	827
Big12	1000	26	25	1000	18	18	2	723
Big13	1000	25	25	1000	19	19	2	738
Big14	1000	25	25	1000	19	19	2	712
Big15	1000	25	25	1000	23	23	1	744
Big16	1000	25	10	1000	22	22	4	208
Big17	1000	25	10	1200	36	30	5	85
Big18	1000	25	10	1000	30	30	3	244
Big19	1000	25	10	1000	28	28	3	212
Big20	1000	25	10	1000	30	30	2	363

Table 4.3: 20 Large Problem Instances for Study 2

Instance Name	Courses	Rooms	Features	Students	Average Student/course	Average Course/Student	Average Room Option	Courses With One Room option
Medium1	400	10	10	400	17	17	1	321
Medium2	390	10	10	400	18	17	1	235
Medium3	390	10	10	400	20	20	1	267
Medium4	410	10	9	400	19	20	1	214
Medium5	410	10	9	450	21	20	1	221
Medium6	410	11	10	450	23	21	1	281
Medium7	410	11	10	450	27	25	1	268
Medium8	400	10	10	400	22	22	1	284
Medium9	400	10	10	400	27	27	1	337
Medium10	400	10	8	500	22	17	5	75
Medium11	400	10	8	800	35	17	4	110
Medium12	400	10	8	800	39	19	3	117
Medium13	400	10	8	800	46	23	4	121
Medium14	400	10	8	1000	44	17	4	106
Medium15	425	10	8	500	23	20	4	111
Medium16	400	10	8	1000	70	28	5	66
Medium17	400	10	8	800	48	24	4	117
Medium18	400	10	8	1000	75	30	4	93
Medium19	410	10	8	1000	73	30	4	85
Medium20	410	10	8	1000	73	30	3	127

Table 4.4: 20 Medium Problem Instances for Study 2

Instance Name	Courses	Rooms	Features	Students	Average Student/course	Average Course/Student	Average Room Option	Courses With One Room option
Small1	200	5	5	200	17	17	2	45
Small2	210	6	5	400	34	18	2	19
Small3	200	6	5	400	50	25	3	16
Small4	200	5	8	500	47	18	2	42
Small5	200	5	8	500	50	20	2	72
Small6	200	5	3	1000	61	12	2	38
Small7	200	5	3	800	76	19	1	84
Small8	225	5	10	1000	88	20	1	95
Small9	225	5	10	900	100	25	2	130
Small10	220	5	10	1000	113	25	1	144
Small11	200	5	4	1000	81	16	2	61
Small12	225	5	10	1000	55	12	1	159
Small13	225	5	10	1000	88	20	1	146
Small14	225	5	3	1000	88	20	3	24
Small15	200	5	3	900	95	21	2	0
Small16	200	5	3	900	95	21	3	24
Small17	200	5	3	900	135	30	2	29
Small18	225	5	3	1000	111	25	2	35
Small19	225	5	3	1000	124	28	3	40
Small20	225	5	3	1000	75	17	2	39

Table 4.5: 20 Small Problem Instances for Study 2

4.3.2.2 Mechanism for Obtaining Results

In this study, the aim is to find feasible solutions. So, the quality of a solution can be judged by measuring the level of feasibility achieved. If all courses are assigned, there will be zero clashes. If courses remain un-assigned, it means their assignment must be creating clashes. This number of clashes is counted, which tells how far from feasibility the solution is. The algorithms being compared have used the following cost function to measure the total distance-from-feasibility:

$$f = \frac{\sum_{i=1}^{s} (Ci)^2}{s}$$

Where *s* is the target number of timeslots, *Ci* is conflict-degree of timeslot *i*. i.e, for each event in timeslot *i*, the number of unallocated events with which it creates clashes.

4.3.3 Study 3: Effect of using Individual Heuristic Operators

This study investigates the effect of using individual heuristic operators on the allocation capability of the algorithm. The aim is to find out the effect of using different heuristics. Instead of using Heuristic Selector Operator, each heuristic is selected one-by-one manually and results are obtained for all instances. The following heuristics are tested:

- ***MostClashingCourses:*** courses which clash with most number of other courses allocated first.
- ***MostClashingStudents:*** course sharing most number of attending students with other courses allocated first.
- ***MostClashingCourses&Students:*** course with the highest ratio of clashing courses and students allocated first.
- ***MostFeatureNeeded:*** courses needing most number of features allocated first.
- ***MostStudents:*** course having most number of attending students allocated first.

- ***MostFeasibleRoom:*** rooms which are feasible for most number of courses allocated first.

4.3.3.1 Problem Instances

The same instances from Study-2 are used in this study.

4.3.3.2 Mechanism for Obtaining Results

The same cost function as in Study-2 is used to measure the total distance-from-feasibility. Value of the cost function is measured within iterations after specified time intervals for each of the problem instances to obtain the results.

Chapter 5: Discussion of Results and Future Work

5.1 Study 1

5.1.1 Results

Solution Quality (Number of problems remaining)							
Instance	**FHO**	**GB HH**	**VNS**	**TS HH**	**LS**	**AA**	**DCABA**
Small1	10	6	***0***	1	8	1	5
Small2	9	7	***0***	2	11	3	5
Small3	7	3	***0***	***0***	8	1	3
Small4	17	3	***0***	1	7	1	3
Small5	7	4	***0***	***0***	5	***0***	***0***
Med1	243	372	242	***146***	199	195	176
Med2	325	419	***161***	173	202	184	***154***
Med3	249	359	265	267	-	***248***	***191***
Med4	285	348	181	169	177	***164.5***	***148***
Med5	***132***	171	151	303	-	219.5	166
Large	1138	1068	-	1166	-	***851.5***	***798***

Table 5.1: Results for Study 1

5.1.2 Legend

- A smaller number means better result, zero means perfect solution, no value denotes that the allocation is not achieved.
- Bold figures represent previous best solutions.
- Bold and highlighted figures represent new best solutions.
- Abbreviations:
    - FHO: Fuzzy Heuristic Ordering by Asmuni and Burke
    - GB-HH: Graph-Based Hyper Heuristic by Rong Qu and Burke
    - VNS: Variable Neighborhood Search by Abdullah and Burke
    - TS-HH: Tabu-search Hyper Heuristic by Kendall, Soubeiga and Burke
    - LS: Local search by Socha
    - AA: Ant Algorithm by Socha

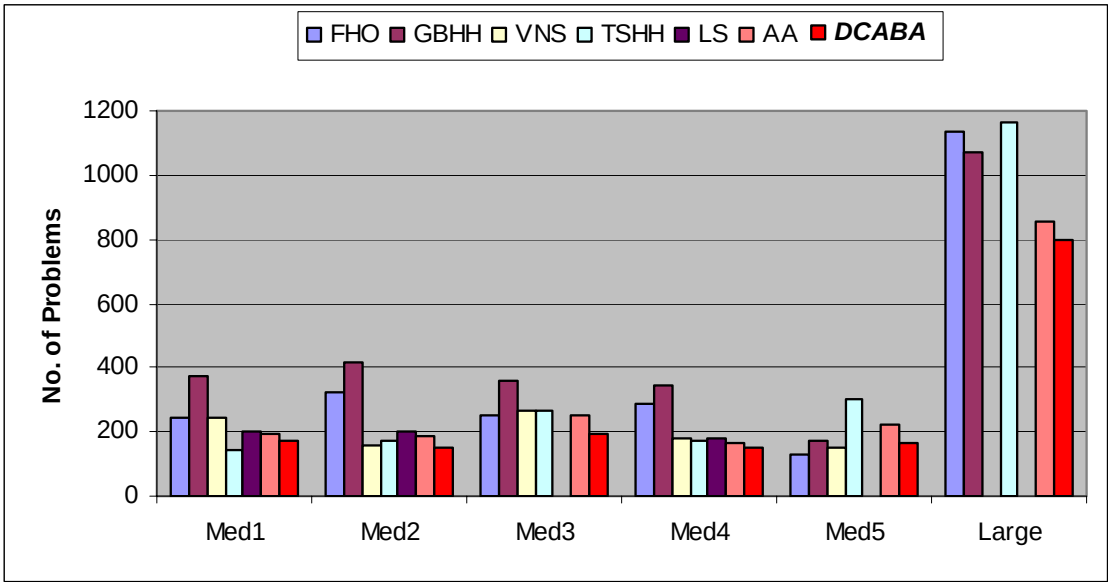


Figure 5.1: Results for Study 1

5.1.3 Findings

- Allocation has been achieved on all the instances and better results are achieved on 5 instances.
- DCABA gives better results on medium and large instances, where problem size is large, with respect to number of courses and students. This is inline with the nature of the helper and orientation changing mechanism in phase 2, which have more chance of success where time space is larger and it has more options to shuffle and swap the courses.
- Results show a strong allocation capability, as in all instances, allocation has been achieved and enough time is available for optimization phase. This finding is further investigated in Study-2.

5.2 Study 2

5.2.1 Results

Solution Quality (Distance-to-feasibility)									
Instance #	Small Instances			Medium Instances			Large Instances		
	GGA	HSA	DCABA	GGA	HSA	DCABA	GGA	HSA	DCABA
1	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	30	8	***0***
5	0	0	0	0	0	0	24	30	***9***
6	0	0	0	0	0	0	***71***	77	78
7	0	0	0	34	***14***	31	145	150	***0***
8	4	***0***	3	9	0	0	30	5	***0***
9	***0***	***0***	7	17	***2***	17	18	3	***0***
10	0	0	0	0	0	0	32	24	***10***
11	0	0	0	0	0	0	37	22	***0***
12	0	0	0	0	0	0	0	0	0
13	***0***	***0***	5	3	***0***	3	10	0	0
14	3	***0***	11	0	0	0	0	0	0
15	0	0	0	0	0	0	98	***0***	37
16	0	0	0	30	***1***	34	100	***19***	38
17	0	0	0	0	0	0	243	***163***	263
18	0	0	0	***0***	***0***	61	173	164	***132***
19	***0***	***0***	9	***0***	***0***	49	253	232	***223***
20	0	0	0	***0***	3	47	165	***149***	159

Table 5.2: Results for Study 2

5.2.2 Legend

- A smaller number means better result, zero means feasible solution, any value greater than zero means allocation is not achieved.

- Bold figures represent previous best solutions.
- Bold and highlighted figures represent new best solutions.
- Abbreviations:
    - HSA: Heuristic Search Algorithm by Rhydian Lewis and Ben Paechter
    - GGA: Grouping Genetic Algorithm by Rhydian Lewis and Ben Paechter

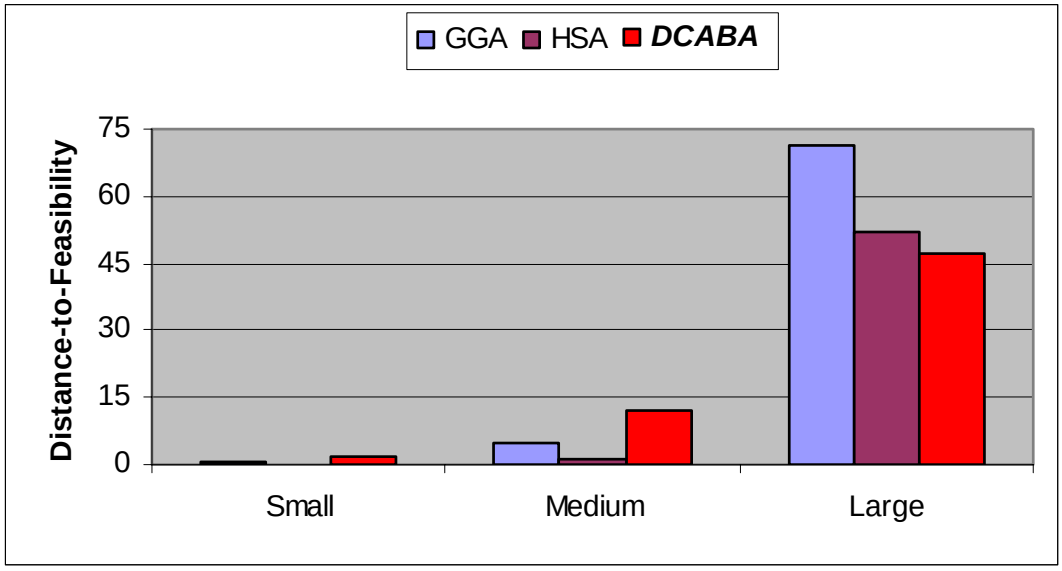


Figure 5.2: Results for Study 2

5.2.3 Findings

- DCABA has given comparable results on most of the small and medium instances. Better results are achieved on large instances, where a better solution has been achieved on 9 large instances.
- This supports the finding from Study-1. In allocation phase, DCABA has also given better results where problem size has been large, with respect to number of courses and students.

5.3 Study 3

5.3.1 Overall Results

Graph shows number of times each heuristic provides best result in term of reaching close to feasibility

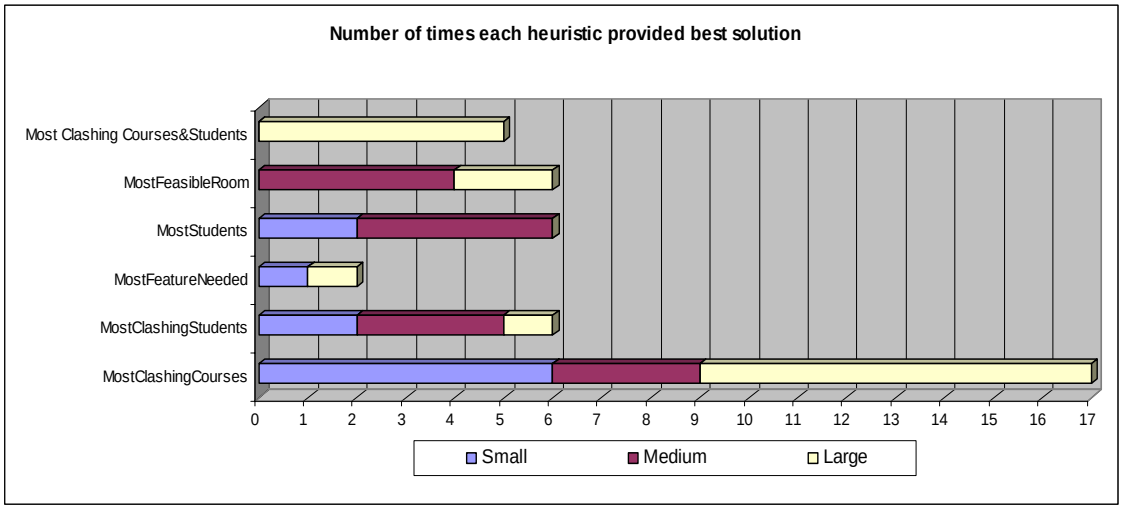


Figure 5.3: Results Summary for Study 3

5.3.2 Findings (Overall)

- MostClashingCourse has given best results 17 times.

5.3.3 Results of Large Instances

Graph shows average distance-from-feasibility on large Instances.

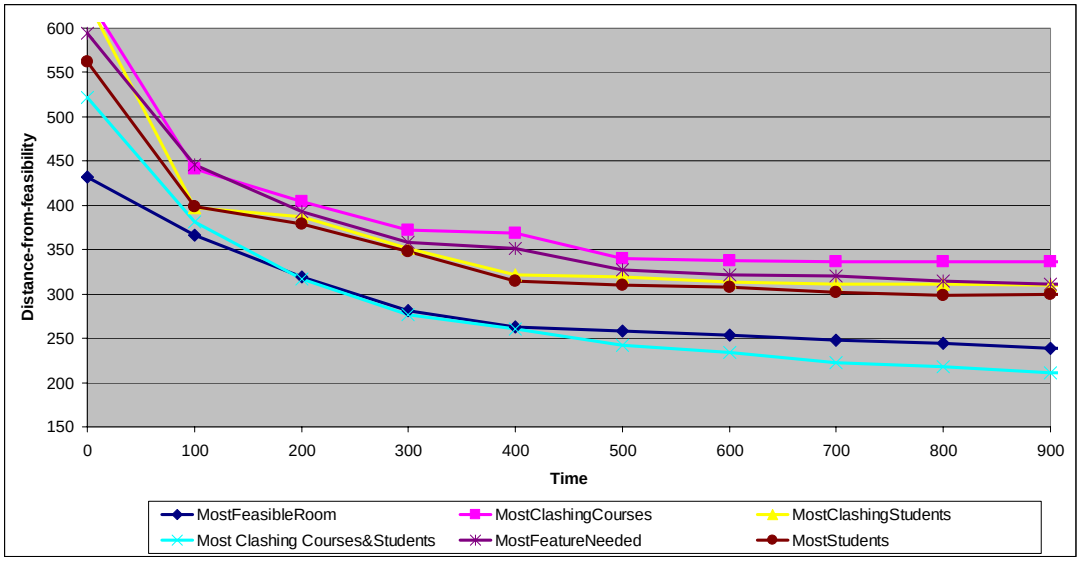


Figure 5.4: Results for Study 2 (Large Instances)

5.3.4 Findings (Large Instances)

- *MostClashingCourse* overall gives most of the best result but has given worst result here.
- Initially, *MostFeasibleRoom* started well but later *MostClashingC&S* has given best result on large instances. This is inline with the characteristics of large instances (more courses, more students, more features needed, less feasible rooms, thus more conflicts).
- A combination of both these Room & Course selection heuristics gives better results.

5.3.5 Results of Medium Instances

Graph shows average distance-from-feasibility on medium Instances.

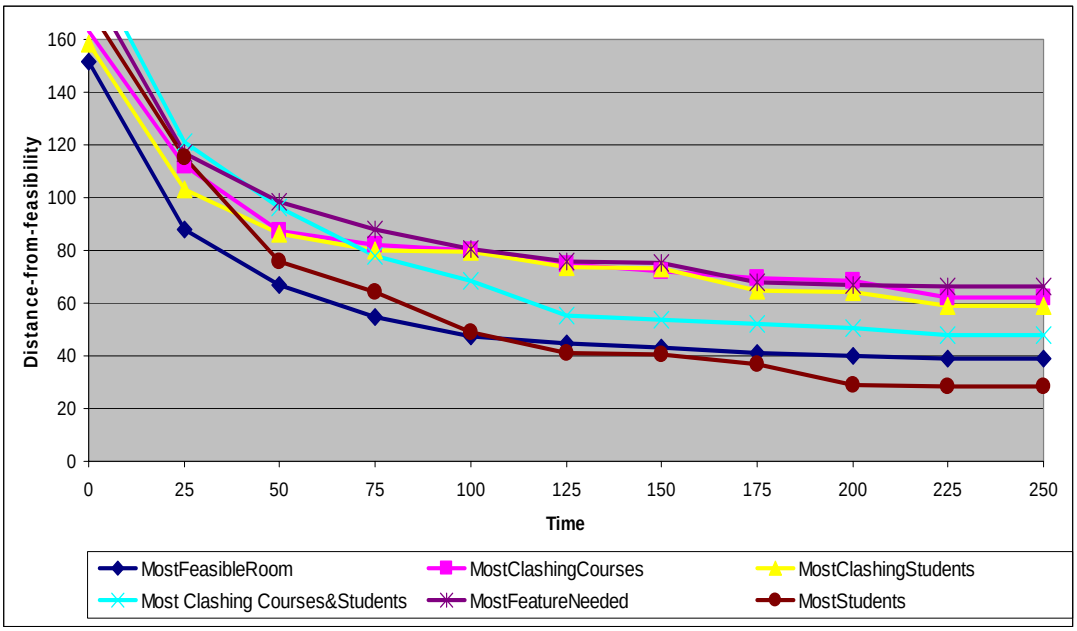


Figure 5.5: Results for Study 2 (Medium Instances)

5.3.6 Findings (Medium Instances)

- No Conflict avoidance heuristic (which select next candidate based on conflict-degree. e.g. *MostClashingStudents*, *MostClashingCourses&Students*, *MostClashingCourses*) has performed well here. The reason seems to be the less conflicting nature of instances.
- *MostStudents* & *MostFeasibleRoom* has given best results. i.e, these heuristics have successfully focused on less no. of rooms available.

5.3.7 Results of Small Instances

Graph shows average distance-from-feasibility on small Instances.

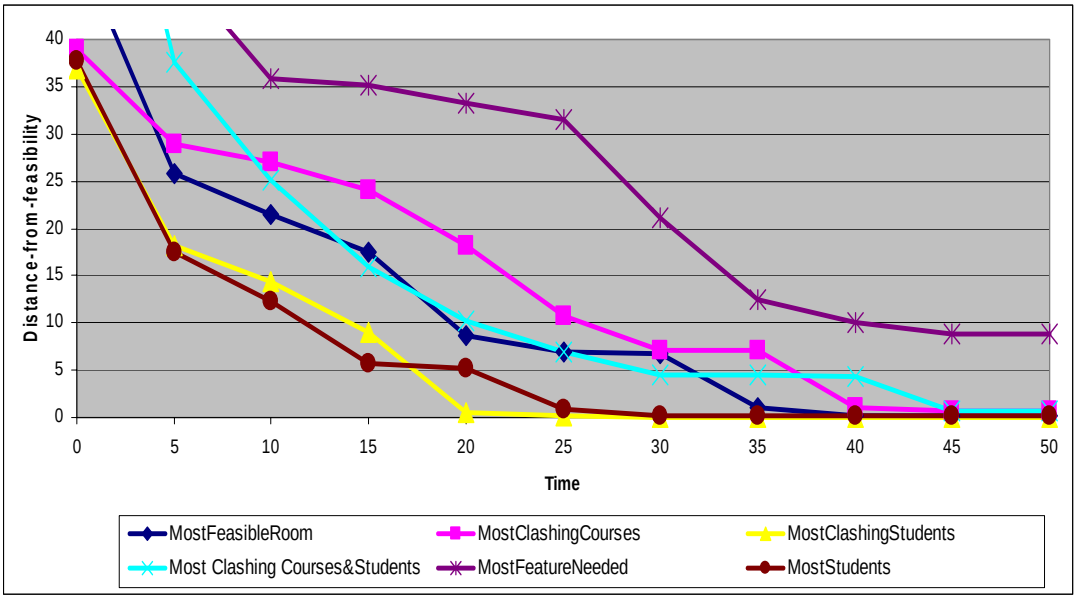


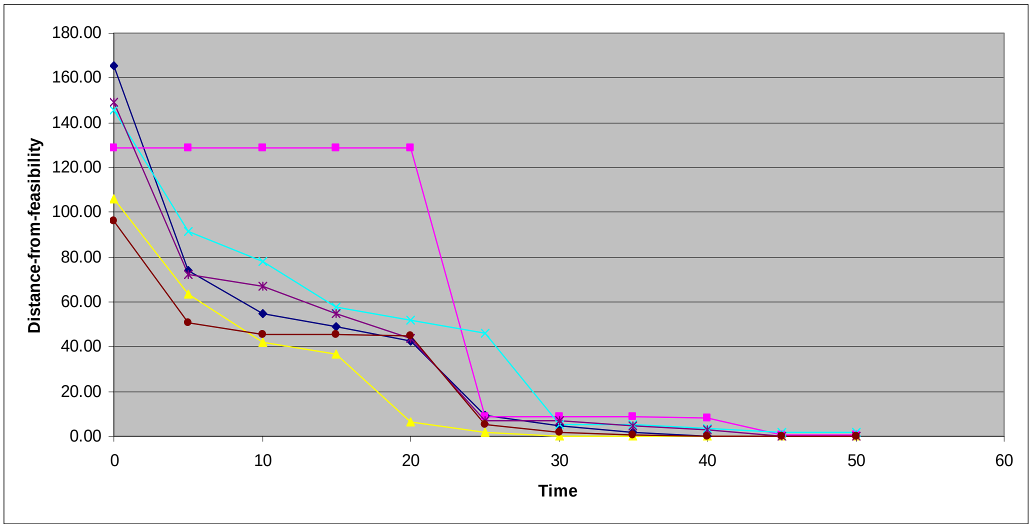
Figure 5.6: Results for Study 2 (Small Instances)

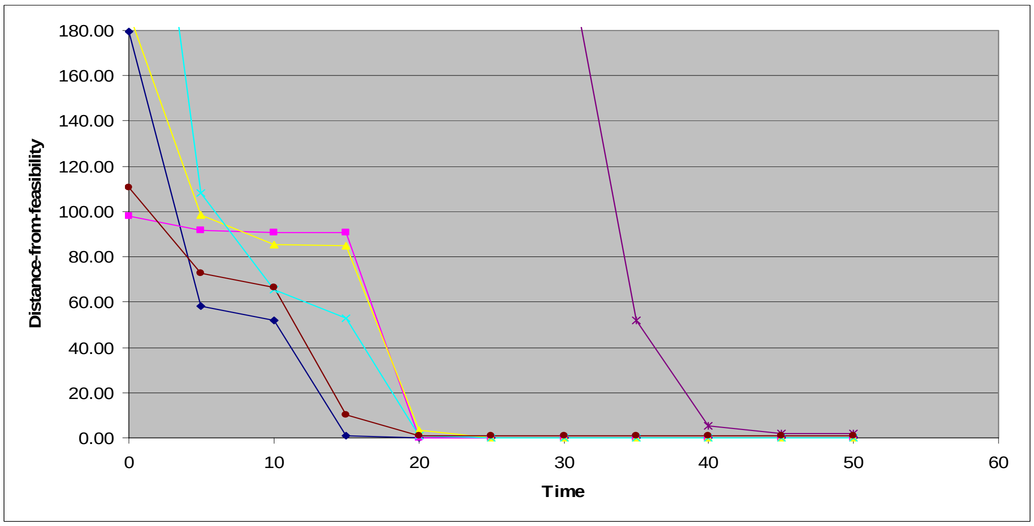
5.3.8 Findings (Small Instances)

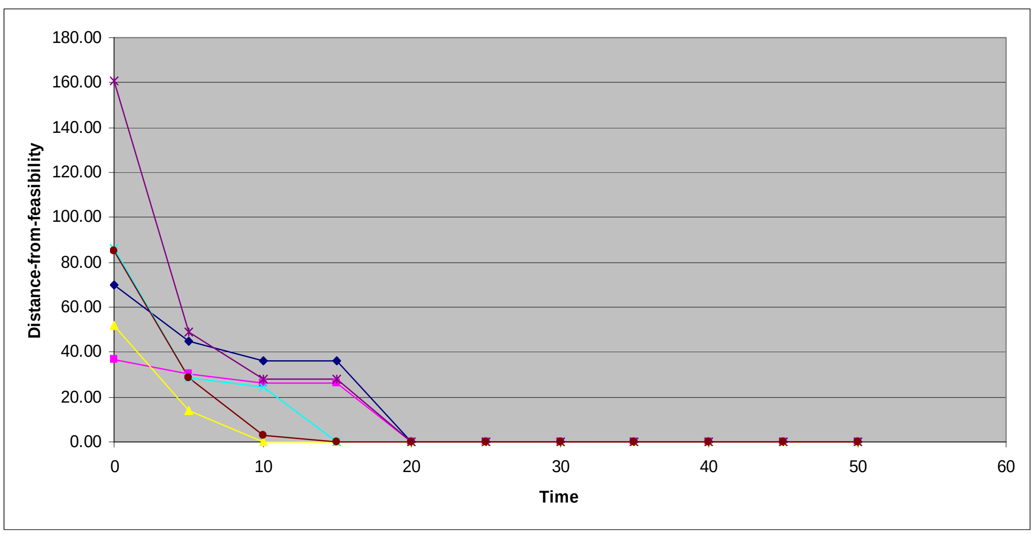
- *MostClashingStudents* & *MostStudents* have performed well in these instances.
- Like the large instances, a combination of these heuristics has given good results.
- *FeaturesNeeded* has been the worst, as feature requirements were negligible in small instances.

5.3.9 Random Behaviors

Some graphs to show the random behavior of heuristics in small instance.







5.4 Conclusions

- This work has been presented as an attempt to provide a diverse approach towards event scheduling. The presented approach has shown promising results. The approach is stronger in the allocation phase than in optimization phase. It has been noted that in most of the instances, allocation was completed easily.
- It has been clearly seen that different heuristics have potential to provide good result in some scenario. Instead of using one heuristic throughout execution, using heuristics suitable for specific scenarios arising while execution may give better results. Instance-specific heuristics, however, have shown potential when they are defined more specifically in accordance to the instance characteristics. Further, instead of relying totally on conflict avoidance heuristic, as in general practice, definition of some more elaborate heuristics clearly shows improvement.
- A combination of heuristics, both for selection of candidate event to be assigned and selection of timeslot gives better results.
- It is a clear finding that selection of heuristics plays an important role on the quality of solution but at the same time, no generalization is observed in this regard either in this thesis or in literature, which identifies the factors responsible for different performance, while using the same algorithm on instances having similar characteristics.

5.5 Future Work

In the following areas, there is potential work for future:

- Improvement of helper and orientation-changing mechanisms.
- Effects of using helper and orientation-changing mechanisms of different capabilities.
- Effects of using different combination of the heuristics.
- Hybridization with other approaches. e.g., fusion with the ant approaches already in research for a broader ant-based framework.
- Modeling this approach for the solution of other assignment and combinatorial optimization problems like Traveling Salesman problem, Minimum Spanning Tree problem, Job Shop scheduling, Vehicle Route Scheduling problems and Bin-packing problem.
- Studying the reason why specific algorithms perform better than others, to identify the factors involved in different results obtained using same algorithm on instances with similar characteristics.
- Attempting an intelligent meta-heuristic algorithm, which analyzes the scenario while execution and select the appropriate heuristic to use.