The Airline Time Tabling Problem: Use of Perturbation and Parallel Search Techniques

THESIS

Submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

by

M Sasikumar

Under the supervision

of Dr S Ramani, NCST, Mumbai

Birla Institute of Technology and Science

Pilani (Rajasthan), India

2003

Acknowledgements

The first note of gratitude goes to Dr Ramani, my thesis supervisor for his keen interest in the subject, his persistence, encouragement and guidance, and for finding time whenever required despite his hectic schedule. Next my gratitude goes to NCST as an institution, for a wonderful working environment, for allowing me to pursue my studies along with my work, and for allowing usage of the resources.

I also acknowledge the many helpful interactions with various colleagues in Air India. These interactions, apart from providing the inspiration for taking up this problem for investigation, helped to understand the issues involved and to gather realistic data for our studies.

I am grateful to all my colleagues in the AI and Educational Technology groups (for which I carry the major responsibility) for their support and encouragement. Undertaking intensive research work of this nature along with management responsibilities in an organization proved to be a major challenge. I am sure the prayers of my colleagues, friends and family have helped a great deal in getting me to this stage, today.

I am also grateful to the staff at BITS, Pilani for their meticulous style of work, help and support at all the stages during the programme.

M Sasikumar

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE
PILANI, RAJASTHAN

CERTIFICATE

This is to certify that the thesis entitled "***The Airline Time Tabling Problem: Use of Perturbation and Parallel Search Techniques***" and submitted by ***M Sasikumar*** ID No. ***1995PHXF401*** for award of Ph.D. Degree of the Institute, embodies original work done by him under my supervision.

Signature in full of the Supervisor Srinivasan Ramani.

Name in capital block letters: ***Dr S RAMANI***

Designation ***Former Director, National Centre for Software Technology***

Date: May 2, 2003

Abstract

Preparing an airline timetable is a complex scheduling problem of great practical significance. At the same time, this is also a problem of high computational complexity. The very large search space and the numerous types of constraints make an exhaustive search for solution infeasible. There are no algorithms fast enough for solving the problem; we will show that no such algorithms are possible.

This problem forms part of the class of vehicle scheduling problems. It is also representative of many types of operational planning problems at the enterprise level in many industries. The economic value of running airlines efficiently cannot be ignored, even if solving the problem requires heavy computing effort. The fact that heavy computing activity involving parallel search can be carried out inexpensively using clusters of computers points to one approach to the problem.

The most critical requirement of the solutions produced to this problem is that the time tables produced should be feasible; that is, no hard constraints should be violated in the time table.

Finding the ideal schedule is not the primary concern. Our objective is to use computing resources to gain a return higher than the cost of that effort. The higher the benefit we get, the better; but we don't necessarily have to have the best.

The solution mechanism also should make it possible to use earlier time tables (apart from being able to start fresh runs) as a starting point since their use usually simplifies the task substantially. Time table preparation cannot be fully automated, as problem parameters are changed several times by the managers as they see different versions of the time table coming out of the planning process. Creation of a satisfactory time table is thus an iterative process involving many

discussions and compromises among the different parties involved. Very fast computing is necessary to enable human schedulers to try out various alternative sets of parameters. For instance, they may try increasing the number of aircrafts in the pool of resources, through proposed leasing. This would be a step forward if the gains are higher than the cost of leasing.

The domain knowledge to be incorporated in some manner in the computational model is substantial. A number of constraints, ranging from bilateral agreements between countries to airport constraints such as permitted take off and landing hours, decide the feasibility of the time tables considered.

We investigate the perturbation-based approach for solving the above-mentioned airline time tabling problem. In this approach, we start with a given schedule and modify it by applying various perturbation operators. The model we use shares many characteristics of genetic algorithms, involving exploration of multiple perturbations in parallel, starting from a set of initial schedules.

We start with a formulation of the problem and then analyze its computational complexity and the nature of its search space. The analysis includes a formal proof for the NP-completeness of airline time tabling problem.

A number of issues have to be addressed before using a perturbation-model based approach to solving a problem. These include choice of perturbation operators, size of the population of candidate solutions, and population management. We study these factors and their effect on the efficiency of the computation. Other issues discussed and/or reported in this thesis include a framework for characterizing perturbation models for this class of problems, formulation of notions such as retention strategy and persistence factor and their systematic investigation, proposal and investigation of a cooperative scheduling

model among competing airlines and investigation of the use of distributed computing model for speeding up computation.

The thesis also offers the formulation of a simplified version of the airline time tabling problem as a benchmark for this class of problems for further investigation, keeping in mind that we should not miss out the essential features of the problem while simplifying it. Simple models have been of great value in other areas of research, as in the case of the drosophila model in biology. A number of aspects related to computational models for the airline time tabling problem are also addressed, including the design of a powerful load-model.

Table of Contents

Table of Figures

Chapter 1. Introduction

Scheduling, broadly speaking, refers to the process of assigning resources to various requirements. Requirements for scheduling arises in many domains including:

a. educational institutions: classroom scheduling, examination scheduling, faculty assignment, etc.

b. factories: job shop scheduling, maintenance scheduling, worker scheduling, etc.

c. transportation: railway time tabling, city route planning, airline time tabling, crew scheduling, tail-number assignment, supply/distribution scheduling, etc.

d. resource management: nurse scheduling (rostering), shift scheduling of workers, scheduling of critical and expensive resources such as operation theatre, radio telescope, etc.

These are all problems of high practical significance with many person-months invested on a regular basis for working out the schedules. The cost of using a poor schedule is extremely high in most of these domains – measured in terms of monetary cost, inconvenience to people concerned, wastage of resources (including human), and so on. This realization has resulted in much attention being given to various scheduling problems even in the early days of computing. However, scheduling is known to be a complex problem in terms of modeling all the domain parameters and constraints as well as in terms of performance requirements.

[Fang, 1994] summarizes the reasons for the complexity of scheduling problems into four categories:

1. Many scheduling problems are known to be NP-complete and hence have no practically feasible algorithms.

2. Heuristics do not guarantee optimal solutions. And most heuristics do not even ensure near optimal solutions. Finding effective heuristics is a difficult task.
3. There are plenty of problem dependent details, which need to be incorporated into the solution method.
4. Real world problem instances contain many constraints, which are hard to specify clearly or represent. Rule based systems have been used to capture the insight of human scheduling experts. This normally requires hundreds of rules. Purely algorithmic formulations are unable to cope with a high percentage of these rules.

Various classes of scheduling problems have been addressed by researchers over the years with techniques ranging from traditional OR models to AI approaches. These problem classes include job-shop scheduling, classroom time tabling, examination scheduling, gate assignment at airports, many variants of vehicle routing and/or scheduling, etc.

[Jain and Meeran, 1998] characterises the various solution approaches along different dimensions: optimization vs approximation, and constructive vs iterative. They observe that optimization methods are effective only for trivial problem sizes and they are too brittle with respect to the domain constraints that can be handled. The major approaches can be categorised as

1. mathematical formulations including mixed integer programming (MIP), Lagrangian relaxation and goal programming.
2. branch and bound models
3. approximation models including priority dispatch rules, heuristics focusing on bottlenecks, constraint satisfaction models, and neural networks

4. local search and meta-heuristics including iterative improvement, simulated annhealing, genetic algorithms, and tabu search.

Though the discussion in the paper is specifically about job-shop scheduling, the categories and issues are equally applicable to scheduling problems in general. The authors observe that the only practically useful approaches are approximation and local search models. Even with them, many challenges are still remaining to obtain satisfactory solution models for solving scheduling problems of real-life magnitude and complexity.

Bulk of the work in solving scheduling problems have been either very formal considering massively simplified problems, or specialized to developing models which suits a particular problem without adequate effort to understand the generality or characteristics of the approach followed. This is particularly true of approximation models. For example, though there have been substantial amount of research on using genetic algorithm (GA) model for solving scheduling problems, there are still hardly any guidelines available on how to model a GA for solving a given problem, or even to decide if GA will be suitable for solving that problem. Most work, given the flexibility in the GA model, resorts to designing specialized crossover operators, chromosome representations, parameter choices, etc which works best for the selected problem. Thus scheduling continues to be a field open for much original research in search of deeper insights into the suitability of various solution models against the various characteristics of the problem.

Airline time tabling is a strategically important and computationally challenging problem in the category of scheduling problems. This chapter outlines the motivation for selecting this problem for investigation in this thesis, in terms of modeling and computational challenges involved. The chapter also explains the problem domain in some detail, and then outlines the thesis plan.

1.1 Computing Power and Computational Complexity

A look at the history of computing would easily show that the pace at which the computing power (measured in, say, MIPS) has been increasing over the years and the pace at which the cost of computing has been decreasing, have been very high. Despite warnings of a slow down in the growth of computing power on a single chip, the growth has so far continued. In addition, there have been tremendous developments in the area of parallel and distributed computing [Sasikumar et al, 2000] over the last few decades. This enables the use of multiple computing units (multiple CPUs in a single computer as in the case of shared memory computing, or multiple computers connected by a LAN/WAN as in the case of distributed computing) to solve a given computing problem.

Low cost supercomputing is possible today with the ideas of cluster computing becoming practical. The term cluster computing refers to viewing a set of stand-alone computers interconnected by a good LAN – the typical configuration often found in many offices and student labs – as a parallel computing environment. Compared to a multiprocessor, these kinds of environments are normally used for coarser grain parallelism models due to the relatively low communication speed among the processing units, compared to a multiprocessor machine. Off-the-shelf packages are now freely available for setting up cluster environments [Sterling et al, 1999]. Among the world's fastest computing setups today, many are cluster-computing environments. Massive cluster computing environments deploying thousands of PC/workstation class computers are achieving performance ratings nearing Teraflops.

Thus harnessing a large amount of computing power is not a major concern today, thereby enabling one to conquer more and more complex problems. A good example for this is the Deep-Blue machine from IBM performing at Grand Master level in chess. Its strength came

primarily from the ability to scan a very large segment of a vast search space using plain brute force exploration – this segment was much larger than what could be attempted using traditional single computer setups. This approach opens up new possibilities for addressing computationally challenging problems.

Many of these problems are believed to have only exponential complexity algorithms (unless P = NP can be proved), and therefore, an exhaustive enumeration of the search space is infeasible. With exhaustive search, additional computing power may yield only fractional improvements in performance, given the vastness of the search space. However, there are non-systematic exploration methods becoming highly practical, which may provide a mechanism for exploiting such computing power effectively. These methods include simulated anhealing, genetic algorithms, and perturbation models.

We could quantify the computing power available in terms of what we call a MIPS-year. A MIPS-year is the computing effort put in by a one MIPS (million instructions per second) computer working for one year continuously. Assuming that a server of 20,000 MIPS costs around one million Rs per year – considering depreciation, maintenance, etc. – we can estimate a MIPS-year to cost about 50 Rs. If an airline's profit (=revenue – expenses) is, say, Rs 50 billion, and if we could offer an improvement of say 5% in the profit by use of optimization techniques, the savings is in the range of Rs 2.50 billion. It is not unreasonable to expect a fraction of this – say Rs 0.20 billion – to be spent on obtaining computing resources for the optimization task. Based on our computation, for this price, we can get computing power of a few million MIPS – equivalent to a few teraflops - or alternatively a specialized supercomputing setup with a capacity that is a fraction of a teraflop.

One of the aims of the thesis is to show the significance of this finding in that, applying a large computing power using suitable problem

solving model can be a very cost effective way to achieve overall improvement in profits. Use of distributed computing strengthens this approach further by offering higher computing power, at much lower cost than that of buying a specialized supercomputer.

1.2 Airline Time Tabling Problem

Airlines plan their schedule of operations a few months in advance and prepare time tables outlining the routes, timing, frequency, type of aircraft expected to be deployed, etc. The periodicity of this time tabling work may vary from airline to airline, ranging from 3 to 6 months. Preparation of airline time tables is a complex process involving many constraints and parameters. At the same time, it is a problem of high economic significance. With the cost of an aircraft running into billions of Rupees, and with operating costs and profits of individual flights being of the order of millions of Rupees, even small improvements in operating efficiency translates to substantial savings even in relatively small airlines.

A special attraction of this problem is that it is representative of a class of problems – the ones central to the conduct of an enterprise, involving a whole lot of information relevant to various departments of the enterprise. These are not isolated, easily sovable problems. Finding solutions to these problems make the very difference between making a profit or a loss, running into billions of rupees. An important feature of these problems is that they involve large revenues and large expenditures. A small percentage change in one of them can result in a big change in the profit or loss, which is only the difference between two large numbers.

As a matter of fact, the airline industry is a rich domain for resource scheduling problems. There are a number of different types of scheduling [Rushmeier et al, 1995] required in course of an airline's operation. These include:

a. The time tabling problem mentioned above, wherein the schedule of operations of the airlines, to be publicly announced, is prepared well in advance.

b. Crew scheduling (also known as crew rostering) for assigning specific crew members to duty on specific flights.

c. Tail number assignment, which assigns specific physical aircraft to specific flights (the time table assigns only notional or logical aircraft numbers to flights).

d. Gate assignment for assigning gates in an airport for arriving and departing aircrafts.

e. Maintenance scheduling for various types of maintenance requirements of an aircraft.

f. Dynamic scheduling, which is necessary to plan responses to unexpected events that occur, such as technical faults, flight operations being disrupted by weather, etc.

While these are all inter-related to some extent, for practical reasons, airlines generally handle them as separate problems.

An aspect that makes airline time tabling problem (ATP) complex is the involvement of multiple departments in the process. Airline operations involve many functional departments including marketing, operations, maintenance/engineering, and crew management. The optimization perspectives of these departments vary, and often may be in conflict with one another. Engineering would prefer the aircraft to be on ground long enough for it to conveniently schedule maintenance and regular engineering checks, without undue pressure on the engineers and resources. The marketing department would like to keep the ground time to a minimum to increase revenue. Crews impose their own constraints on the frequency and timing of flights. As discussed earlier, assignment of a specific crew and a specific aircraft identified by its tail number cannot be done when the time table is prepared. Since

availability of crew and the constraints in scheduling them for duty on flights imposes constraints on the feasibility and viability of time table, logical numbers for available teams are used during the time tabling stages, as a via-media approach. Which actual team is identified for handling a particular flight is decided only a few days before the flight, mapping a logical crew number into an actual crew number. Similarly, the identification of a specific aircraft (identified by its tail number) against a logical aircraft number is done close to the actual time of the flight.

Perhaps the most significant of all the difficulties in preparing an airline time table is preparing a good demand model. For scheduling and time tabling problems to be solved effectively, we require a model of the expected demand (number and type of passengers and their preferences). In general, for any given airline, there are a number of competing airlines with significant overlap in operating regions. Therefore, it is nearly impossible to estimate a realistic demand (i.e., available load in terms of numbers of available passengers to different destinations, categorized by the class of travel) for every region for a given airline. The passengers also can differ in their preferences. The various relevant groups include business travelers, holiday travelers, group travelers, short-trip travelers, long-trip travelers, etc. The passenger profile can affect the preferred timing of flight, tolerance to stop-overs and delays, type of aircraft used, etc.

The problem is also computationally very complex given the number of parameters to be considered. For example, [Rushmeier et al, 1995] reports that USAir's time tabling involved around 2500 flight legs, about 400 aircrafts belonging to 14 different types and 140 airports. Numeric models for this problem can run into millions of variables/constraints.

Given all this background, one can see that the airline time tabling problem is a high-value problem offering many challenges to the

human schedule makers. In this thesis, we focus on this problem, analyze this in detail from various perspectives, and develop a solution-framework based on perturbation. The questions raised in this process and our answers to them are discussed in Chapter 4 onwards.

1.3 The Problem Formulation

This section defines the problem of airline time tabling in detail, identifying the major inputs and constraints. The information here has been collected from close interaction with a specific airline, and moderated based on the studies in literature [FSAS, 1999][Rushmeier et al, 1995].

Task: Prepare a weekly time table for an international airline. Such a time table lists all flight routes along with the arrival and departure times at each station en-route and the type of aircraft to be used. The objective is to maximize the net profit, satisfying all the hard-constraints and as many of the soft constraints as possible.

Inputs: There are a number of inputs to the system as follows.

1. Airport information: name, time offset from GMT, curfew timings, slot times, etc. Related information such as inter-port distances and block-times (flying-times between two airports) is also required.
2. Aircraft information: name, type, seating capacities for different classes of passengers, range (that is, the distance it can fly non-stop), speed and fuel consumption per hour.
3. Route information: A route is a sequence of airports. A flight operating on such a route stops for passenger pickup/offload at all the airports in the route. For a given airline, there will be a set of such routes. We assume that each scheduled flight operates along one of these routes. In the simplest case, we can name each airport-pair as a route.

   Routes often represent a variety of constraints and thumb-rules

evolved over years of field experience. In the case of airline time tabling, routes codify information such as over-flying and commercial rights in various countries, generally-expected load across sectors, and refueling opportunities and fuel charges at various airports. Such information is fairly static over a reasonably long period of time and hence the resulting thumb rules need not be discovered every time a time table is to be prepared. Use of abstractions such as routes, provide a convenient mechanism to exploit human expertise without losing the ability to make modifications as and when required. In Chapter 3, we discuss the rationale and significance of routes in more detail.

4. Demand model: We assume this to be of the form, <from_loc, to_loc, type, number, preferred day, preferred time> indicating the number of passengers of the specified type wanting to travel from "from_loc" to "to_loc". Since we are operating on a weekly schedule, the preferred day indicates if the specific passenger group has a preference for a particular day of the week. The issue of suitable demand models is discussed in detail in Chapter 4. The input data also includes the expected revenue per passenger for each class, for every pair of source-destination locations.

5. Right to load/discharge passengers at any airport (to/from any other airport): This is governed by the agreement between the countries involved. We do not consider this as a separate input, merging it with the demand model. We assume that the demand model lists only those from-to combinations, where the airline has a right to transport passengers from the "from" location to the "to" location.

6. Various charges applicable: A variety of charges are paid by the airlines to various agencies for operating a flight. These include landing charges (for each landing), parking charges (per hour), over-flying charges (for countries that the aircraft flies over during a flight), fuel charges, passenger amenities, etc. Many of these

charges are also dependent on the type of aircraft used, and in some cases also on the time of operation. Hence it is a complex task to create accurate estimates for such charges.

Constraints: The major constraints that govern time table preparation are listed below. Some of these result in additional data inputs to the system, whereas some of the constraints are taken care of by suitable assumptions on the inputs listed earlier.

1. Aircrafts should not land or takeoff at any airport during the curfew timings, if any, specified for the airport.
2. Aircraft cannot fly non-stop over distances greater than the specified range. The range parameter in the aircraft data gives the desired data. Compliance with this constraint must be ensured while making schedule.
3. The schedule must be cyclic across the week. That is, the aircraft position at the end of the week must match that at the beginning of the week. This is basically to ensure that there is continuity between time table for one week and the time table for the next. It is important to note that there may be no time during the schedule-period where all aircrafts are located at a common base. Indeed, at any given time, one or more aircrafts may be in the air en-route on a flight. Therefore, the system cannot start the schedule assuming a common location for all the aircrafts at time=0.
4. Passengers should not be lifted/discharged at any place where the airline has no right to do so. We assume that demand model lists only valid location-pairs. If a larger common data source such as the statistics from IATA (International Air Travel Authority) is used to start with, pre-processing should produce a pruned demand-list including only those location-pairs where the airline has the right to load and discharge passengers.

5. Passengers prefer routes with minimal stopovers. This is a soft-constraint.
6. Aircrafts leaving the country will usually choose a return route to return to the originating country, once it reaches its initial destination. That is, after every return flight, the aircraft will be within the originating country. In our model of the airline time tabling problem, we assume that specific route-patterns as described earlier will be given as input, capturing a variety of information on what is allowed and what is not. To meet this constraint, we stipulate that every route either starts within the country or terminates within the country. Relaxing this requirement on the route will relax this constraint as well.
7. Respect passenger preferences (for desired timings for departure/arrival, for example) wherever available – another soft constraint.
8. A specified minimum amount of time must be spent at each airport en-route during a flight for cleaning, engineering checks and other ground handling activities. In some cases, this may include some types of maintenance checks also.

Assumptions

a) We assume that long-term maintenance is not an issue during time tabling. We expect the aircraft count given as input to take this into account and give the effective number of aircrafts that will be available at any point in time. Thus we do not schedule *all* the physical aircrafts during the time tabling process. If the airline has 10 aircrafts of a specific type, the engineering department would not offer, say, more than 8 of them to be in operation simultaneously, two being down for various reasons including various types of maintenance. We use internal aircraft identification such as v1, v2, ..., v8 to denote these and work with

these "logical" aircrafts. The logical aircraft v1 may correspond to different physical aircrafts during different weeks. As mentioned earlier, this problem of mapping from logical to physical aircrafts is called the tail-number-assignment problem.

b) We have ignored crew constraints. As mentioned earlier, detailed crew scheduling is not carried out during the time tabling process. Some simplified model is normally used – for example, the maximum number of crew sets assumed to be available – to ensure that the schedules produced do not put undue demands on crew scheduling.

1.4 Time Table Representation

We define a general structure for the time table for ease of discussion in the subsequent chapters. We also use the following conventions for referring to the major input items:

V = set of vehicles (in this case, the aircrafts), v_i denotes a specific vehicle.

L = set of locations (in this case, airports), l_i denotes a specific location.

R = set of routes, r_i denotes a specific route.

Time table = {<v, vehicleschedule> | v ∈V}
vehicleschedule denotes the schedule of one aircraft.

vehicleschedule = <$Flight_{r1}$, $Flight_{r2}$, $Flight_{r3}$,...>* r1,r2,r3,... ∈ R;
a sequence of zero or more flights, one flight corresponds to one route. More than one flight may be based on one route[1].

---

[1] For example, there may be five flights in a week on the route Mumbai-London-Newyork.

$Flight_r = \langle E_1, E_2, .., E_n \rangle$ where $r = \langle l_0, l_1, l_2, ..., l_n \rangle$ and

E_i denotes the event[2] of flying from l_{i-1} to l_i, non-stop.

1.5 Organisation of the Thesis

In this chapter we have introduced the significance of the airline time tabling problem and described the problem in some detail. In the next chapter (Chapter 2), we discuss literature related to this kind of problems as well as those concerning possible solution strategies. Chapter 3 discusses the airline time tabling problem in more detail, examining the major issues in modeling it. Chapter 4 discusses one specific aspect of airline time tabling, namely the formulation of a load model. Chapter 5 proposes a simplified model of the airline time tabling problem, examines the complexity of ATP and reports some studies carried out to understand the nature of its search space. A number of interesting results are reported.

Chapter 6 discusses the perturbation model based solution for ATP in detail – examing the major ingredients of such a model and the options available. Chapter 7 analyses the model, and identifies the issues taken up for exploration in this thesis, and proposes hypotheses related to these. The experiments conducted using the model to study these issues and the results obtained are presented in Chapter 8. The results are shown graphically and observations are made regarding the usability, effectiveness and efficiency of the model. Chapter 9 discusses distributed computing techniques relevant for exploitation of parallelism in speeding up the perturbation search. We conclude the thesis in Chapter 10, discussing the study, summarizing the contributions of the thesis and identifying possible further work in this area.

---

[2] Information associated with an event includes arrival and departure times for the two airports involved and the type of aircraft.

Appendix A provides programming details about the implementation listing the various modules. Appendix B provides sample data files and outputs produced by the system. Specialised terms used in the thesis are explained in a glossary.

1.6 Contributions of the Thesis

1. Empirical and analytic study of the airline time tabling problem.
2. Development of a general time tabling model for airline time tabling, including a powerful load-model.
3. Creation of a perturbation model for a practically significant problem in a form that can exploit the parallelism of a multi-computer environment.
4. Study of various parameters relevant to the perturbation approach, including perturbation operators, population size, and population management strategy with respect to their effectiveness in solving the airline time tabling problem.
5. Design of a benchmark scheduling problem based on airline time tabling.

Chapter 2. Literature Review

Planning and scheduling problems have been topics of interest, for analysis and formalisation, for many decades, even before the advent of computers. Numerical models as practiced in the Operations Research [Gillet, 1976][Wagner, 1982] field continue to be popular tools for modeling scheduling problems. These models include linear programming, non-linear programming, integer programming, relaxation models and goal programming. Since most non-trivial problems in the area of scheduling have been found to be computationally intractable, a lot of work has also been reported using heuristic techniques [Bodin et al, 1983]. The problems addressed span a wide spectrum varying from traveling salesman problem to multi-depot vehicle scheduling problems and classroom scheduling problems.

From the initial days itself, planning has been an integral topic in the AI research parlance [Georgeff, 1988]. It was viewed as an important application domain even in early AI textbooks [Nilsson, 1980][Rich and Knight, 1982]. Planning [Allen et al, 1990] is concerned with the general problem of deciding an appropriate sequence of actions to achieve a given set of goals. The actions may include management of resources. In these models, temporal issues such as specific time periods for activities and dependencies of time periods on activity and results were largely ignored. With the advent of Mark Fox's ISIS [Fox, 1987] system, there has been an enormous amount of interest [Zweben and Fox, 1994] in scheduling problems from the AI community. A number of technologies have been explored for an equally vast range of scheduling problems.

Thus the field of interest to us contains two different perspectives – of planning and scheduling – and two different technologies – numerical and AI methods. Our problem is closer to the scheduling problems, than general purpose planning problems. Among the scheduling

problems discussed in the literature, time tabling problems are of particular interest to our work, since our task can be seen as a type of time tabling.

Baker defines scheduling [Baker, 1974] as the "allocation of resources over time to perform a collection of tasks". Time tabling on the other hand, refers to a set of events (examinations, classes, etc) being arranged into a set of timeslots subject to domain constraints. We can see that airline time tabling does not belong strictly to either of these categories, but shares some aspects of both. So, in this chapter, we will examine selected literature relevant to formulating and solving time tabling and scheduling problems.

After discussing the general models and concerns, we will examine the perturbation-based model in some detail, since this thesis focuses on that model.

2.1 General Background

There is hardly any work seen in the literature specifically addressing airline time tabling problem. However, scheduling problems, in general, have received a fair amount of attention in the literature [Zweben and Fox, 1994] over the last few decades. Even before the advent of computers, these have been problems of interest, given the cost benefits that can be obtained from even small improvements in the schedule employed. After computers have been deployed to solve complex problems, the interest in these problems has increased significantly [Arabeyre et al, 1969] [Foxley and Lockyer, 1968]. Many algorithms for handling scheduling problems of realistic complexity require exponential effort (that is, the time required to find an optimal schedule grows exponentially with the number of aircraft, number of locations, etc.), and hence not useful for solving real-life scheduling problems. Most of the surveys on scheduling problems (For example, [Bodin et al, 1983] for vehicle routing and scheduling, and [Jain and

Meeran, 1998] for Job shop scheduling) focus only on approximation models, with the view that algorithmic optimization models are infeasible. Much of the work reported in the literature is, thus, concerned with heuristic methods.

Some of the major scheduling problems addressed in the literature include:

1) Job-shop scheduling: A set of jobs have to be assigned processing stations and time-slots for completion. Each job needs to go through a sequence of processes, and has associated dead-lines and priority levels. [Zweben and Fox, 1994] [Jain and Meeran, 1998]

2) Vehicle Routing and Scheduling problems: In these problems, a set of vehicles have to be dispatched to a set of locations to pickup/drop load subject to various constraints. [Cordeau et al, 1998][Bodin, 1990] [Bodin et al, 1983]

3) Crew scheduling and rostering [Arabeyre et al, 1969] [Caprara et al, 1998] [Chu et al, 1995]

4) Classroom time tabling [Frangouli et al, 1995] [Henz and Wurtz, 1995]

5) Examination time tabling [Carter, 1986] [Carter and Laporte, 1996]

6) Gate assignment at airports: This addresses the problem of choosing suitable gates at large airports for arriving and departing aircrafts considering physical compatibility, connecting flights, etc. [Brazil and Swigger, 1988]

7) Railway scheduling [Brannlund et al, 1998] [Chiang and Hau, 1995]

Given the difficulty of solving such problems optimally [Jain and Meeran, 1998], there are two distinct approaches being followed:

i) Exact solution to an approximation of the problem: Here, some of the complex constraints which make the problem intractable are dropped or simplified, and the resulting problem is solved using exact solution methods such as linear programming, integer programming, etc. The resulting solutions are, often, not implementable in practice, because significant constraints are ignored. Therefore, this approach is used either when the ignored constraints are not very critical for implementing the schedule or for getting a good base from which to start the manual or other approaches to scheduling.

ii) Approximate solution to the exact problem: In this line of work, the problem is attempted with all the constraints and specifications in tact. Instead, optimization is compromised. The focus shifts to obtaining as "good" a solution as possible, that is feasible to be implemented. Almost all the current work focuses on this category.

A variety of techniques have been reported for solving scheduling problems. These can be categorized based on different criteria. Considering the approach to schedule-development, we can broadly categorise the work into two classes: *constructive* scheduling and *repair-based* scheduling. We can also categorise the scheduling literature based on the nature of the technique employed to solve the problem. There are primarily three groups: AI approaches, OR approaches and other approaches. We discuss these classes in some detail below.

Attempts have been made by commercial outfits to develop intelligent support environments for airline time tabling and associated tasks. Oasis [OASIS, 1999] is a tool in this category. It provides a rich interactive environment for airline staff to develop time tables, with the system taking care of book keeping and data management. The system supports a wide range of constraints and provides interface to recognized international standards for various data items (for example,

IATA tables for estimating the demand). FSAS – Flight Schedule Automation System – [FSAS, 1999] is another commercial product with similar functionality. It may be noted that none of these systems attempts automated schedule preparation!

2.1.1 Constructive and Repair-based Approaches

In constructive methods, the solution is constructed incrementally whereas repair based approach starts with a complete, possibly infeasible and/or poor schedule and tries to adjust the schedule to correct constraint violations and/or to improve the quality of the schedule. We illustrate these two approaches, considering the N-queens problem.

A constructive approach would start with an empty board, pick up the first queen and place it at some square, say, in the first row of the first column. Then try to place the second queen in the first row of the second column. A violation of constraint would be noticed, leading to backtracking. We change the position of the second queen to second row and then to third row where an acceptable (so far) placement would be found. Then we consider the third queen, and so on. The queens are being added one by one to an initially empty board. A complete solution is available only when the program terminates with no more queens remaining to be placed. A variety of search techniques (including heuristic) [Korf, 1988] have been proposed for building a solution optimally in an incremental manner.

A repair-based approach [Minton et al, 1990][Zweben et al, 1994] would, perhaps, start with placing all the 8 queens randomly on the board – typically one queen per column. As one would expect, this would generally result in one or more queens attacking each other. An iterative repair process is now used. It starts by identifying a queen that is badly positioned and changes its position within the column. If the new position improves the total number or gravity of constraint violations, the new position is accepted. This process is repeated till the

solution in hand satisfies all constraints, or no further improvement is possible.

The repair-based approach has the advantage that if some good earlier solution is available, the scheduling can start from that. Particularly in airlines, this is a desirable approach. It leads to better acceptance by the staff, flying crew and passengers, as it avoids major changes from one time table to next. It is not necessary to develop each time table from scratch. One can use the time table deployed in a "similar" earlier time period and start adjusting it to suit any changes in situation compared to what was prevailing at the time of that time table. There is also the strength that essentially at any time, the repair-cycle can be interrupted and a solution is available. The more the time we give to the system, the better the resulting solution would be. However, there is no guarantee that a repair-based approach would give benefits comparable to what the optimal solution would give.

In the early days, much of the work was based on constructive search model. However, incremental construction approaches are being found to be too time intensive to be of practical use for real-life problems. Recent focus is, therefore, on iterative improvement methods [Dorn et al, 1994]. Mark Fox has also reported such a shift away from purely constructive heuristic search techniques in the case of the job-shop scheduling problem [Fox, 1994]. Good performances are being reported for iterative refinement methods for many problems [Minton et al, 1990][Sosic and Gu, 1991][Sosic and Gu, 1994].

2.2 *AI Approaches*

Scheduling problems have attracted some attention from AI from the early days itself, as exemplified by systems such as Nudge [Goldstein and Roberts, 1977]. It became a major area of research in AI, focusing on practical scheduling problems, primarily due to the pioneering work of Mark Fox [Fox and Smith, 1984] in the area of job-shop scheduling.

This work led to an explosive growth in scheduling as an AI problem, spawning work on many different scheduling problems using many techniques.

AI approaches can be further divided into two classes based on the other categorization dimension – constructive AI approaches and repair-based AI approaches.

2.2.1 Constructive AI Methods

Constructive AI methods view time tabling as a state-space search problem [Korf, 1987] [Korf, 1988] where a state is a partial schedule where some flights have been scheduled for some aircrafts. A move is any operator that can expand such a partial schedule. The schedule is constructed incrementally as a sequence of partial schedules. The flights in a state may not necessarily have been done in any particular order. Partially specified flights are also possible in a partial schedule. For example, a flight has been scheduled on a route, but the specific aircraft type has not been fixed for the flight or the starting time has not been fixed.

In using constructive approach for airline time tabling, we may start with an empty schedule, that is, no flight has been formulated for any aircraft. A move from a given state considers the various ways in which the current state can be extended. Mostly this would be by adding a new flight. It can also be by tightening the specification of any partially specified flight in the current state. If a satisfactory move is found, the current partial schedule is revised suitably (by adding the identified new flight or tightening the selected partially specified flight) and a new state is generated. Various choices for adding a flight (which route, which aircraft, which day, what time, etc) lead to different extensions of the current partial schedule. If a satisfactory move is not found from the current state, the current branch is terminated, and the current state defines a possible time table.

The optimization problem is to identify the optimal time table among the feasible ones. A variety of search techniques have been reported for carrying out this procedure. The search space being too huge for exhaustive search, heuristic techniques are generally used. Tabu search [Glover, 1990], beam-search, iterative deepening, etc [Russel and Norvig, 1994] have been tried. Evaluation functions can be defined to be applicable to partial schedules as well, so that among the possible extensions, only the most promising few are retained for further exploration. Constraint programming models to constrain possible options at any stage have also been explored [Hentenryck, 1996].

The constraints can be checked/ensured during selection of valid moves and during the evaluation of the resulting state. Soft constraints are normally handled by assigning penalties for violation during the evaluation of the state.

2.2.2 Iterative Improvement Methods

In iterative improvement methods, each state represents a complete schedule. The moves are perturbations to the schedule. We will use the term perturbation-based approach for this model. The term repair-based signifies that changes are made in the current time table only for correcting constraint violations or inconsistencies. Since we are also interested in optimization, we would like to incorporate changes in the time table if that leads to an improvement in profit.

Taking a classroom-scheduling example, we can randomly generate an assignment of classrooms, teachers and timeslots for every period of interest. Such a random assignment may contain a number of inconsistencies or violation of constraints, let alone have the solution anywhere near being optimal. However, the schedule is complete. This forms a state in this model. A move from a given state performs some modification (perturbation) to the state, and thereby obtains a new state. The moves are generally meant to improve the quality of the schedule as determined by an evaluation function. For example: change

the teacher assigned to period A, change the time slot of subject B, swap the slots assigned to teacher A with that of teacher B, etc. The applicable modification procedures are called perturbation operators.

Perturbation operators can be simple or complex. They are generally problem dependent – primarily due to efficiency considerations. Though, it is conceivable to visualize a generic perturbation model based on the structure of the state representation, these are likely to be undirected and inefficient. Such a generalization is based on the notion that given an adequately specific definition of the state structure, we can identify some generic ways for modifying the structure. For example, if a vector of objects[3] is present in a state, the operators can include addition of a new element to the vector or deletion of an element from it. We can also modify an existing element, which will translate to recursively modifying the components of the element.

Note that though at the broad level we can visualize such perturbation operations, this involves many complications in state representations of real problems. For example, deleting an element from a vector may require changes to be made in the other elements of the vector and in some cases even in other parts of the state. For example, if a flight is deleted from the schedule of one aircraft, the estimated load carried by another flight of a different aircraft may increase. Consider morning flights from location A to location B. If there are P passengers as potential load for this segment, they will be divided among the available flights servicing this segment. If there were two flights operating in this segment, and one is deleted the expected load for the other flight would increase. Similarly, when a new flight is added, the

---

[3] In the case of the Airlines Time tabling Problem (ATP), such an object could represent a series of flights taking an aircraft from a base to another location and bringing it back to a base.

estimated load for the existing flights may reduce. In addition, airlines have the concept of connecting flights, where a passenger traveling from A to C travels via one flight from A to an intermediate location B, and then proceeds via a different flight from B to C. This also adds complexity to propagation of changes in a schedule. We explore these and other such issues in detail later on.

Having described the constructive and perturbation models under the AI framework, let us examine the various techniques employed to realize these models.

2.2.3 Solution Techniques

The major techniques deployed for solving scheduling problems include:

1) heuristic search including hill climbing, branch and bound, constraint-directed beam search [Fox, 1987], etc. alone or in combinations.
2) constraint programming [Frangouli et al, 1995] [Hentenryck, 1996] including constraint logic programming [Cohen, 1990]
3) Genetic algorithms [Schoenauer and Michalewicz, 1997]

Heuristic search was the preferred model during the early days. The first version of ISIS [Fox and Smith, 1984] [Fox, 1994] adopted such a mechanism, using constraints primarily to prune the search tree early. It was soon noticed that constraints are perhaps the most important component of real-life scheduling problems, and hence it was imperative to provide them a more constructive role in the schedule-making process. Subsequent versions of ISIS incorporated these – the approach has been called constraint-directed beam search.

ISIS went further to exploit the constraints, proposing specialized scheduling architectures such as opportunistic scheduling. Using domain knowledge, the scheduling was split into three types depending on what forms a bottleneck during a given state in scheduling. The idea

was to focus the scheduling process on the bottleneck component. Thus, dynamically the system would move between order selection, capacity based scheduling and resource scheduling.

ISIS also pioneered the idea of preparing a general framework for solving job-shop scheduling problems. Constraints were given a clearer structure including its relation to other constraints, alternatives, costs and priority.

Despite the fair success of job-shop scheduling, the complexity of the constructive model followed by ISIS was overwhelming. Lack of adequate frameworks and inadequate understanding of the roles of the different types of constraints in the scheduling process has led to exploration of alternative models. Constraint programming and genetic algorithms were two popular contenders that emerged in the process.

Constraint programming (CP) provided the advantage of focusing on the most critical component of scheduling problems – the constraints. It looked for ways to identify and categorise the different types of constraints, and to provide a general representation and associated inferencing models for them. The essential idea in a CP-based approach is to view the problem as one of assigning values to a set of variables. The set may be finite or infinite, statically known or dynamically determined. The constraints are viewed as restrictions on the values that given variables can take. The constraints can be unary (v1 should be an odd number), binary (v1+v2 > 10), ternary, etc. Powerful constraint propagation models can exploit these constraints to restrict the values that the various variables can take.

In general, the propagation algorithms and value representation mechanisms available today are not powerful enough to exploit the constraints fully. For example, given v1+v2 < 10 where v1 < 10, v2 < 12, we can easily eliminate 10 and 11 from the domain of v2. These constraints, however, convey much more information. To apply further pruning, we need to wait till more information is known about either v1

or v2. Therefore, CP based methods ultimately rely on search techniques (including heuristic search) for deciding on values.

Many of the constraint programming frameworks [Hentenryck, 1994] [Shulte et al, 1998] available today provide sophisticated constraint representation models, propagation algorithms including simultaneous equation solving and numerical algorithms, and powerful search techniques. Scheduling and time tabling has been favourite domains for trying out CP frameworks [Hentenryck, 1994][Frangouli, 1995].

Genetic Algorithms (GA), formulated by J Holland [Holland, 1975] is an evolutionary approach to problem solving. The basic idea is to take a population of potential solutions and let them evolve to "better" solutions. New solutions are derived from those present using a few types of predefined operations. These include crossover, which merges parts of multiple solutions to generate a solution which incorporates aspects from each of the solutions chosen, and mutation which is a change carried out on a single solution at a time. The "poor" solutions – as defined by an evaluation function – are discarded, and the better ones are retained for the next round. The model, though simple, has been found to be very effective in solving a variety of scheduling problems [Fang, 1994].

Though relatively less understood theoretically compared to other models, the empirical results are highly positive. One of the major strengths of the approach is its generality. There are only a few domain specific properties required for formulating a GA-based system for solving a problem. These are:

a) A representation for the solution – GA imposes no constraint on what this should be.

b) An evaluation function mapping a solution to a real number characterizing the fitness of the solution.

c) Crossover and mutation operators, which take the required number of solutions and returns new solutions. Again, the details of how this is done are of no concern to GA.

However, the success of the approach depends on a number of factors, which are hard to determine analytically, and therefore, calls for a fair amount of trial and error [Forrest and Mitchell, 1993][Park and Carter, 1995]. For example, the convergence of the process is dependent on having proper crossover and mutation operators and good evaluation function. Empirical guidelines are being explored for implementing GA based solutions. One of the pioneers of genetic algorithms, DE Goldberg says "many of the first generation evolutionary and genetic algorithms currently in use are incapable of solving hard problems quickly, reliably and accurately" [Goldberg, 1998].

All these techniques have been applied alone or in combination. For example, hill-climbing type of local searches are being used effectively in conjunction with a GA model to improve performance. Similarly many systems apply constraint propagation techniques in their model, even though they may not be using a constraint programming framework for the problem as a whole.

[Zweben et al, 1994] describes a system called GERRY based on perturbation (iterative repair) model. It makes heavy use of a constraint based representation of the domain. Schedule-IT is a system for allocating drivers and vehicles for delivery of orders to customers [Duncan, 1995]. The system is based on constraint-based approach, and was built using ILOG Solver.

2.3 OR Approaches

The literature of Operations Research is rich with scheduling related problems [Bodin et al, 1983][Bodin, 1990] [Rushmeier et al, 1995] [Cordeau, 1998] [Brannlund et al,1998]. In particular, vehicle routing and scheduling problems [Thangiah, 1991] [Magnanti, 1981] are a

topic of considerable study in this area. Bodin [Bodin et al, 1983] discusses a number of variants of the vehicle routing/scheduling problem including:

a) traveling salesman problem

b) Chinese postman problem

c) M-travelling salesman problem

d) single depot, multi vehicle, node routing problem

e) multiple depot, multi vehicle, node routing problem

f) capacitated Chinese postman problem, and so on

Recognising the computational complexity of real-life problems of this category, most of the methods proposed are heuristic in nature. The problems are categorized based on the nature of optimization and constraints. For example, routing (without assigning times) and scheduling are considered as separate problems; single depot and multiple depots for the vehicles lead to another categorization; and whether demands have time-windows or not is yet another dimension. Even with such limitations, most of these are unable to incorporate the many ad-hoc constraints that arise in the domain.

The approaches and heuristics for the various models are substantially different from each other. A variety of models [Gray et al, 1997] have been attempted including linear programming, integer programming, mixed-integer programming, generalized goal programming [Ignizio, 1983] and Lagrangian relaxation models. [Brannlund et al, 1998] describes a railway time tabling system based on integer programming model. [Cordeau, 1998] surveys a number of routing models which employs techniques such as dynamic programming, Lagrangian relaxation and simulated annhealing.

Thus, though a rich literature is available, many admit that plain OR solutions are not adequate for solving real-life scheduling problems.

Another aspect contributing to this issue is the amount of expertise required for formulating problems under an OR framework, and the opaque nature of solution. Getting partial solution or explanation for certain behaviour is almost always impossible in an OR framework [Dhar and Ranganathan, 1990].

2.4 Hybrid Models

Recognising the search complexity of AI models and the OR approach's inability to reflect real-life problems adequately, some attempts have been made to combine the strengths of both. There have also been efforts to bridge the two approaches by encouraging each group to benefit from the strengths of the other. For example, Grant makes strong suggestions in this direction, particularly emphasizing that OR approaches need to make use of domain knowledge and informed search methods for being effective with real-life problems [Grant, 1986]. One simple approach is to use OR generated initial solutions as good starting solutions in perturbation models.

A few studies have also adopted an expert system kind of approach for solving scheduling problems – where one identifies a human expert's way of solving these problems and mimics that by codifying the expert knowledge in rules or any other suitable framework. [Solotorevsky et al, 1994] describes a specification language called RAPS for such a scenario. When such rules are available and optimization is not a major criterion, the approach is a viable option to examine. However, most scheduling problems have an optimizing component and there is no reason to expect that human expert's solutions are even near-optimal. Similarly, many constraints are too complex to be contained in a rule-based model.

Many authors report low-quality results when restricting to a single approach for solving practical problems. [Jain and Meeran, 1998] lists a number of cases, where hybrid models have produced much better

results than what has been accomplished with a single model. These include memetic search [Corne et al, 1999] – a recently popularized search model – which incorporates a local search over a genetic algorithm based core model.

2.5 Comparing the Different Approaches

The above discussion covers a number of different approaches/models for solving problems such as scheduling and time tabling. [ILOG, 1997] is a whitepaper by ILOG Inc providing comparative study of four different approaches for a variety of optimization problems. The paper compares rulebased approach, constraint programming approach, simulated annhealing and genetic algorithms. Rather than recommend one specific approach, the paper concludes that each of the models have their own strength areas and based on the formulated model of the problem, an approach should be chosen. From the lot, genetic algorithms and simulated annhealing are more general purpose and fairly effective for a range of problems. Constraint programming, despite being general, is not effective by itself to produce solutions efficiently, and requires a powerful search model to accompany. [Jain and Meeran, 1998] also has a similar message, observing that most of the approaches in the literature from numeric models to approximation models show problems with scalability and applicability to a general class of problems. There seem to be a general bias now towards iterative refinement models – with local search and genetic algorithms in the lead.

Iterative refinement models offer the following advantages over constructive approaches:

1. There are very few assumptions on the domain or the task. The overall model is applicable, therefore, to a wide variety of tasks and domains.

2. The areas where domain specific information or knowledge is required is well-defined in the overall model – the solution representation, the evaluation function, etc. The rest of the solution framework is independent of the problem.
3. Primarily due to the above characteristic, it is easier to modify the problem specification by adding/deleting constraints and parameters.
4. Since the model works with complete solutions at any time, one can start with existing solutions (perhaps used earlier) and apply iterative refinement. This is particularly useful, if the changes are incremental – fresh effort starting from scratch is not required.

More and more of the work reported for general scheduling problems is using iterative refinement or constraint models. This trend is presumably because of the need to be flexible in specification of constraints in real life scheduling problems. Based on the comparative observations earlier, we have opted to use perturbation model as the basis for studying ATP in this thesis.

2.6 Problem Representation

In general, most of the attempts at solving scheduling problems have been attempted as one of its kind type. Though, one can classify problems broadly as time-tabling, vehicle routing/scheduling, etc., these do not seem to provide adequate generality at the problem representation level to define the structure of a scheduling problem. However, for automatic time tabling, scheduling, etc to move into wider applications in main stream of information technology, powerful tools and models are essential. A case in point is the advent of expert system shells, which contributed significantly to the popularization of expert systems in industry. A few research attempts have tried to develop general models for specific classes of scheduling problems. School time tabling and job-shop scheduling are some of these domains.

A number of different types of scheduling problems have been identified as of general interest. Job-shop scheduling for scheduling jobs on machines on a factory floor is perhaps the most well-known. [Fox 1987] and [Fox and Smith, 1984] report pioneering efforts in formulating it as a general problem class. Another class that has been fairly well understood is school time tabling. Most other scheduling problems lack a model and framework. One of the reasons for this lack of model is perhaps that the domain constraints play a more significant role in deciding solution strategies than the general problem parameters. Addition of certain types of constraints can make certain techniques such as linear programming unsuitable. [Smith, 1992] reports another interesting work attempting to formulate a general scheduling framework for transportation problems. [Solotorevsky et al, 1994] proposes a rule based framework for specifying and solving a range of resource allocation problems. The framework identifies the major ingredients of such a problem as:

1. concepts including activities, resources and priorities
2. rules including the restricting rules and recommending rules
3. strategy for solving which includes forward allocation using priorities and recommending rules, followed by consistency checking for possible violations and taking corrective actions based on what is called "local_change policy".

The framework proposed (RAPS) provides for specifying these in a higher-level framework, and allows use of different problem solving strategies to be used against the same specification. The authors themselves admit that their framework is not powerful for all problems – "RAPS ... have less ambitious goals".

The categories proposed in RAPS are not adequately defined for all scheduling problems, and do not cover the range of parameters of interest. Time is not given any special consideration, for example.

Constraints are absorbed into the rules – and do not become visible for different parts of the solution strategy to exploit them.

There have also been some work in identifying the different types of constraints, which apply to scheduling problems. Trying to formalize the various types of constraints will also help in mapping solution strategies and problem models to characteristics of problems. [Fox, 1987] attempts a classification of constraints in the domain of job-shop scheduling problems, listing them as:

1. organizational goals (e.g. due date requirement, work in progress)
2. physical constraints (e.g. product size)
3. causal restrictions (e.g. operator precedence and resource requirements)
4. resource availability constraints
5. preferences (of machines, operators, etc.)

While some of these categories are general in nature (resource availability, for example), this categorisation is not directly applicable to other domains such as airline time tabling. So far, there is little success in formulating general models for such scheduling problems.

2.7 Perturbation Model in Detail

Since our focus in the thesis is on perturbation-based approach, we now discuss this approach in more detail. The classic hill-climbing [Russel and Norvig, 1994] [Rich and Knight, 1992] can be considered as the core of the perturbation model. In hill-climbing, there is one solution seed, which is continuously being modified based on local information. From the given seed, variants are generated using pre-defined operators. Among the variants generated from a given seed, the one resulting in most improvement in the solution quality is accepted, and the search is continued with this as the seed. Variations on this model are possible. For example, one may not generate all

possible variants from a given seed, before choosing one to follow. However, the basic idea is to explore within a close neighbourhood for a better solution and move to that solution, if found.

Hill-climbing is, of course, subject to many drawbacks [Russel and Norvig, 1994] including local maxima, plateaus and ridges. On a vast search space, the search can also take very long. Perhaps the biggest problem in searching non-monotonic vicious search spaces using local information only is that local maxima would trap the search.

Using multiple seeds provides significant improvement in overcoming these limitations. The seeds are randomly generated and hence can explore different regions of the search space. Multi-seeded hill-climbing is the basis of perturbation models. Genetic algorithms also fall under the same model. GAs move away from pure hill climbing by using the crossover technique to hop around different areas of the search space by combining characteristics of multiple parent solutions. They also do not explore the complete neighbourhood of a solution. GAs have a high stochastic component, making the search non-systematic.

The general structure of a perturbation based solution strategy is as follows:

1. Generate a population, S, of N solutions
2. repeat
3. pool = S, the current population
4. for each element s in the pool
5. Select a perturbation operator
6. Perturb s to obtain s'
7. Add s' to pool
8. Evaluate all the elements in pool
9. Retain N elements considering primarily the cost
10. S = pool
11. until termination condition met

The basic issues in this strategy are

1. Representation of a solution
2. Generation of the initial population
3. Deciding suitable perturbation operators
4. Population management
5. The termination criteria

We will discuss these issues in detail along with the relevant tradeoffs and options in Chapter 7. The performance of a perturbation model depends on good choices of all these as well as a reasonable population size. There have been studies [Fogel, 2000] – empirical and analytical – to understand the impact of these and other parameters on the performance of the algorithm. However, the answers are all partial. [Fogel, 2000], however, analytically substantiates that it is not possible to have a general purpose GA model that will yield good performance for all problems. Empirical studies have been reported for some scheduling problems including examination time tabling, job shop scheduling, etc using GAs.

The major drawback of models such as GA is its purely empirical nature. While the approach has been shown to work well, for a number of practical problems, there is yet no satisfactory insight into its effectiveness. Most of the applications have developed specialized operators, chromosomes (i.e. solution representation), etc and have also adopted other techniques such as local search as supplements. These make it difficult to know the extent of contribution by the GA approach per se. These also make it difficult to generalize the approach or model to other problems.

The main theoretical result available so far is the schema-based *building block hypothesis* [Goldberg, 1989][Fogel, 2000]. This is applicable where chromosomes are represented as bit strings, and the classical crossover and mutation are used as the only operators. A

schema is a bit pattern over the chromosome. For example, if the chromosome is 6 bits long, 11*011, *****0 and 1****1 are some of the schemas. A 1 or 0 in the schema matches with the respective symbol in a chromosome, and a * matches either (wild-char). Thus a schema containing atleast one star, matches multiple chromosomes. In a population of size N, each individual chromosome will represent one or more schema. Similarly, any given schema may be represented by zero or more individuals in the population. Assume that the optimal solution can be described as a schema in the above notation. The task of the GA can be visualized as locating the right individual(s), which characterizes the best schema.

The building block hypothesis postulates that GA encourages the growth of good schema provided they are short. By favouring better chromosomes to survive, we are encouraging good schema to have larger and larger representation in the population. With finite population sizes, such an approach can result in premature saturation with the entire population becoming representatives of a schema representing a local optima. Mutation as an operator is meant to protect against this, since it introduces random shifts in select chromosomes. Crossover facilitates exchange of ideas among the population members, by supporting merging of chromosomes representing two or more different solutions. This is a powerful operator in the traditional GA model. For crossover to be effective, the optimal schema should consist of relatively independent segments which can be independently discovered and integrated. For example, if the optimal schema is 11**01*, and f is a profit function (i.e., goal is to maximize f) then the following relations must hold:

f(11**01*) > f(11*****) > f(*******)

f(11**01*) > f(****01*) > f(*******)

If partially acquiring the schema components worsens the evaluation, then such partial solutions may not survive, making effective crossover

of such partial solutions towards obtaining the optimal solution unlikely. The field of deceptive functions studies this phenomenon in detail.

The analytical studies of such models are still active areas of research. An important aspect of this study is an analysis of the fitness landscapes, their categorization and their impacts on the GA performance. [Mitchell et al, 1991] describes a hand-crafted function – named the Royal Road function – for studying these kinds of issues.

In this thesis, we do not restrict our attention to GA type of perturbation. However, thanks to the close similarity in the approach, many of the issues and techniques will be common to GA and the general perturbation based models.

Chapter 3. Model for Airline Time Tabling

In Chapter 1, we defined the airline time tabling problem (ATP) and specified its overall structure. In this chapter, we look at some of the unique characteristics of this problem and examine ways to handle them in modeling ATP.

3.1 Basic Entities and Characteristics

The basic components in an ATP are aircrafts, airports and passengers. A significant indirect entity is the route denoting any valid flying pattern.

Each type of aircraft has a number of associated parameters and constraints. These include the range (the maximum distance the aircraft can fly non-stop), the seating-capacity (further broken down into capacity for different passenger classes), fuel consumption per hour, maintenance requirements, speed of flying, etc. A soft constraint is the preference of certain aircraft to be flown in certain sectors (often a marketing consideration).

Airports have similarly a number of direct and indirect constraints. The direct constraints include curfew timings during which no aircraft will be allowed to take-off or land at that airport, slot-timing – the assigned time for specific airlines to land/take-off, and night-navigation facility – without which flights cannot operate at that airport during night. Indirect constraints arise partly from the charges applicable. Landing, takeoff, parking, ground support, etc are all charged at every airport. The charges can depend on the type of aircraft and sometimes on the time of the event. For example, landing at night may be costlier than landing during day, at some airports. Parking beyond a permissible period can attract surcharges as well.

Profit, defined as revenue minus costs, is the primary target for an airlines' operation. However, unlike in the case of most other scheduling problems, estimating the revenue is complex for airline time tabling. The main reason is the complexity of estimating the number of passengers in the various flights in the schedule. The way a scheduling system considers passenger related inputs, is called its load model. We discuss this in depth in the next chapter.

Mostly the flight patterns of airlines can be divided into two models: hub and spoke model and point-to-point model. Many airlines use a combination of these in their schedule. In hub and spoke model, a few airports are selected as hub locations, and flights are run from other airports to and from these hubs. Direct flights among the various airports are minimized. To go from A to B, you may need to go from A to a hub location H and then catch another flight from H to B. The number of hubs one visits en-route can be zero or more (zero being when there is a direct flight). Point to point models chart routes based on traffic pattern and operate direct flights between most locations. Multi-hop flights are common in this model when catering to low-demand sectors. The intermediate points here should not be confused with a hub; they are only a mechanism to share the load on different sectors, and not intended to be transition points.

In the case of hub and spoke model, route patterns will be of the form H-A-H where A is any airport and H a hub. For the point to point model, any pair of airports A-B can be considered as a route. Even in the point to point case, every combination of airports is not necessarily operated directly by an airline. Some are operated as connecting flights, where two flights A-B and B-C are scheduled such a way that some one wanting to go to C from A, can take the first flight, alight at B, change over to the second flight and go to C. The flight departure times are synchronized keeping this in mind. Some flights are also operated as

multi-hop routes (e.g. The route A-B-C can take care of demands across sectors AB, BC and AC).

Using this idea, we can integrate both the flight patterns – hub and spoke, as well as point to point – using the concept of routes. The set of routes given as input will be dependent on the model followed by the airline. The notion of routes is general enough to support both the models.

Routes specify such flying patterns in use, taking into account by revenue considerations, international agreements, etc. Though, it is conceivable to throw open every possible airport combination as a possible route and let the system decide which are worthwhile to operate on, this is too resource consuming to do in practice. The constraints governing the routes are often outside the system purview, e.g., bi-lateral agreements. In addition, these routes are normally fixed by an airline and need not be "invented" every-time the scheduling system is run. Thus, we find the concept of routes as an appropriate way to model these constraints and requirements.

Also note that, such a mechanism allows the human scheduler to guide the system, if required. By providing more patterns or removing select patterns, the system's exploration can be controlled substantially. Though this option should be exercised with care, it does offer a good deal of control for the human scheduler, when required.

3.2 Other Constraints

There are a number of constraints that govern the feasibility and viability of a time table. These have been briefly described in Chapter 1.

Some of these are indirect constraints, in the sense that, we can take care of these constraints by suitable assumptions or restrictions on the inputs used in the system. For example, if we have no right to pickup passengers from airport X, the demand table (load model) can be

pruned of entries listing passenger load from X. We can, then, ignore such constraints during the scheduling process.

Some constraints are preferential. They need not be enforced every time and the system should exercise flexibility to over-rule these constraints, if required. An example of this kind of constraint would be the timing preference of passengers on various sectors.

Other constraints have to be satisfied by any valid schedule. Examples are minimum waiting time at each stopover and aircraft range constraints on the non-stop flying time.

Constraints on crew are hard to incorporate directly in time tabling. Normally crew planning is done after the time table is finalized. Mapping available crew to a time table is itself a complex scheduling problem [Caprara et al, 1988] [Chu et al, 1995]. On the other hand, it is not possible to neglect crew restrictions completely during the time table preparation. Doing so may produce time tables making infeasible demands on the crew, making the cost calculations meaningless.

The overall cost of the schedule is dependent on the crew available. Airlines have agreements indicating the maximum period of time that a crew will work continuously and a minimum rest period whenever this limit is reached or exceeded. Consider a schedule with only one flight a week from A to B, and assume that the flying time from A to B is around the scheduled duty-time limit for crew. In such a case, the crew that went to B on the onward flight, cannot take up the assignment of the return flight, if the return flight leaves soon from B. Either two sets of crew would be required for operating this route with one set positioned at B or the return flight should be delayed sufficiently to allow the crew of the onward-flight adequate rest time. However, if you add one more flight, the same two crew sets can operate that additional flight as well, reducing the incremental cost substantially.

Exploiting all these kinds of constraints completely during the time tabling process, make the time tabling process unreasonably complex. Therefore, some broad indicative parameters (e.g. number of crew sets) only are considered during the time tabling process.

3.3 Load Model

The load model is, perhaps, the most important component of airline time tabling. At the same time, it is the most difficult information to obtain and formulate. A number of issues govern the formulation of a load model. We discuss the formulation of a load-model in detail in the next chapter.

3.4 Revenue and Cost Computation

The cost and revenue parameters in airline time tabling are quite involved. Costs are incurred on a multitude of aspects ranging from crew and fuel, to landing and parking at various airports. In order to make a general model for airline time tabling, we need to organize these costs, so that addition of new types of costs becomes possible without substantial changes to the system.

The costs of airline operations can be broadly classified into two types: the charges associated with being at an airport and the charges associated with flying time. We will term these as ***event charges*** and ***link charges***. Event charges include airport charges, landing charges, parking charges, and surcharges related to these. Link charges are charges during a non-stop flight from one airport to another. These include fuel charges and charges for passenger amenities.

This gives us a simple representation that is general enough to handle all kinds of charges applicable. In our model, there are two files corresponding to these charges: event_charge.ini and link_charge.ini.

The event charge specification has the form:

<location,aircraft-type,arrival-time,departure-time,charge,description>

This indicates that "charge" amount is payable every time an aircraft of the type specified is at the specified airport, satisfying the arrival/departure time constraints. The arrival and departure times allow for conditions such as "> 10:00 hrs", so that we can specify different charges depending on the time of arrival (if required). The description field is primarily to capture the type of charge for internal record keeping and reference (e.g. landing charge). This model allows for easy modification of charges, and can also incorporate new types of charges easily. The main types of charges included at present are the airport charges and landing charges, both of which depend on the aircraft type and the airport.

The link charges can be of two further types in terms of how they are calculated. Some of the charges are dependent on the number of passengers present and the others are dependent on the distance traveled (or alternately the time taken for the flight). Charges for passenger amenities belong to the former set and the fuel charges is of the later type. The link charge specification has the form:

<location1, location2, aircraft type, charge-type, charge, description>

Location1 and location2 indicate that the charge is applicable while flying from location1 to location2. Charge type is one of 'per pax', 'per km', 'per hour', and 'fixed'.

We find this structure adequate to capture all the types of charges we have come across in this problem domain.

3.5 Representing Constraints

Constraints are represented in the system depending on the category to which the constraint belongs (see Section 3.2 of this chapter):

- Indirect constraints are handled by assuming that the data files are suitably filtered before running the time tabling system.

- Preference constraints are applied during the evaluation of the schedule and appropriate penalty factors are assigned. Certain changes of preference constraints (addition or modification) will require changes to be made in the program. Note that deletion of constraints is not a problem, since we can set the appropriate penalty factor to zero.
- Other constraints are enforced at the time of generating solutions and also while making modifications to the solution. These constraints are attached with the classes representing the relevant entities (e.g. aircrafts, airports, etc), and all relevant methods enforce and verify adherence to the applicable constraints.

3.6 Soft Constraints and Perturbation Model

In most optimization models, the hard constraints are handled by enforcing them during the construction of the solution or by imposing a very high penalty factor. Soft constraints, on the other hand, are handled by incorporating additional terms in the evaluation function. Each constraint violation is assigned a penalty and each type of constraint is given a weight; the weighted sum of all these penalties forms a part of the evaluation function. The difficulty in this approach is in deciding relative weightages among the various soft constraints, making them comparable to the other terms in the evaluation function (for example, compare satisfying passenger time preference with reducing the number of stopovers in a route). The problem is more complex in constructive scheduling models, since deciding these factors in a partial schedule do not, in general, reflect the scenario that arises when the schedule is completed.

Perturbation models, and iterative improvement models in general, provide a more useful approach to address this problem. Particularly, in domains such as time tabling, many of the soft constraints can be

modeled differently, in a more effective way. Mostly soft constraints in the airline time tabling domain, has a direct/indirect impact on the passenger traffic and the revenue. The fact that we always deal with a full schedule enables this to be exploited effectively in our approach.

For example, let us consider the soft constraint of minimizing the number of stopovers en route. What is the rationale to have more stopovers? The airlines can cater to load to/fro the intermediate locations in addition to the load between the end-points. A direct flight from A to C can cater to the load from A to C only; but if the flight is routed via B, then it can also cater to the A to B and B to C loads. This may help me to increase the load factor on the flight. What is the rationale to reduce stop over? Passengers prefer to spend less time in travel. Also aircrafts take longer to finish the flight whereas it could have been deployed on another sector.

All these can be directly modeled in perturbation models. Firstly, the incorporation of additional load that the multi-hop flight brings will be visible when computing the estimate of the load on the flight. Aircraft taking longer to finish a flight is bad only if there are other good opportunities available for it to do. This will be visible when comparing this schedule with other schedules in the population. Note that a direct estimate of this opportunity cost is very involved to compute.

The human preference factor can be handled by a heuristic function which converts the estimated number of passengers available for the A-C sector into a number likely to board the specific flight under consideration. Anyway, we need such a function to take care of load drop off due to timing preferences at arrival and departure (see next Chapter), type of aircraft, etc. This function can also build in a factor to reduce the number suitably based on the number of stopovers.

Similar approaches can be formulated for most soft constraints in a multi-seeded iterative refinement model, such as the one we are employing. This strength primarily comes from two factors. Firstly we

are always dealing with a full schedule and hence opportunities and alternatives for passengers are easy to see. Secondly development of the solution progress by comparing the various solutions with one another, enabling opportunity costs to be modeled easily in an indirect manner.

3.7 Analysis of the Model

The cost representation model is at present a direct representation of the file structure mentioned earlier. To reduce heavy disk I/O, the files are read into an internal buffer and accessed. No other optimizations have been incorporated. The access to these structures, in general, can be significantly speeded up, reducing the net run time significantly (though at the constant term level only). Possible optimization measures include:

1. Indexing of the entries based on airport to reduce search time.
2. Use of better structures for holding the data.
3. Pre-processing of the general table to a more efficient format. This can be done as a separate stand-alone phase.

The model is, otherwise, general enough to represent real-life airline time-tabling problems.

In the implementation, we have incorporated directly only some of the constraints since the focus is on studying the problem in general, rather than catering to a specific airline's requirement. The model can easily absorb further constraints.

For implementation, we have followed an object-oriented model. Most of the entities discussed here, are represented directly by classes. All hard constraints are handled by the classes (see Appendix A) corresponding to the entities on which the constraint applies. Addition of new hard constraints may necessitate changes in the code corresponding to the relevant entities; however the interface projected

by the various classes would remain the same, and hence other classes need not be affected.

When the system is considered for use in real applications, the use of a database to hold most of the data can also be considered. For example, it is expensive to use text-files to represent large statistical databases such as those obtained from IATA. The availability of commercial database systems providing general information such as travel times, is another factor in this direction of thinking. Our representation does enable a database structure to be incorporated easily.

Chapter 4. Load Model for Airline Time Tabling

Once a flight is planned, we need to know how occupied that flight is likely to be, and what kind of revenue it is likely to generate. Passengers and cargo[4] are the only profit-enhancing elements because they pay; everything else increases the cost of implementing the schedule. The load model is the description and distribution of passengers (and cargo, in the general case) who need to travel from place to place. In this chapter, we discuss the issues in formulating such a load model, discuss and analyse our proposed load model.

4.1 Parameters Affecting the Load Model

The basic requirement in formulating a load model is to find out how many passengers are interested in traveling from a given location A to another location B in a week. A major difficulty in obtaining this information accurately is the likely presence of competitor airlines covering the same sector. It is not easy to estimate how many of the potential passengers will use our flight vis-à-vis the other airlines' flights.

Passengers choose flights based on flight-specific parameters such as the number of stops en-route, departure time at source and arrival time at destination, in addition to the cost of ticket. There is also, often a loyalty factor, further complicated by schemes such as frequent flier schemes.

Number of passengers carried by an airline in the previous year, is often used as a base for demand profile. However, this is obviously biased by the flight pattern in force at that time. For example, if the

---

[4] In this thesis, we have considered only passenger load models and have not considered cargo.

airline did not operate a flight in the sector A-B in the previous year, and is considering a flight in that sector now, there is no reliable demand information one can gather. Even more significantly, the airline may have had a flight from A to B, say at morning 9.00 o'clock. The number and type of passengers on this flight do not necessarily give us a complete picture of the potential load in the sector. Given the many types of passengers and their constraints/preferences, the load carried may change if you change the flight timing. Thus, the available data cannot be easily extrapolated to predict the load if the flight time is changed to, say, 6.00pm.

Thus, one will not have a truly reliable demand profile for airline time tabling. Previous years' load pattern, modified by inputs from marketing experts and field staff, is often used for this purpose. The input from the marketing department plays a crucial role in the formulation of demand model. Their surveys, intuition, study of competitor airlines and so on should contribute to the final design of the load model.

Impact of timing is a major concern in airline time tabling. For example, a business traveler will prefer to spend as little time in travel as possible and will have less flexibility in changing flights. Their preferred timing would be to arrive at the destination in the late evening to have a night's sleep before getting to work or arrive early in the morning to have a full day ahead of them at the destination. Family and tourist passengers prefer to optimize use of holidays and weekends in choosing flights. For example, for Air India and Indian Airlines, one of the major sources of passenger traffic is Gulf countries. The passengers are primarily those who have gone for work there and coming to India on leave or returning after holidays. They prefer flights matching the weekly-holiday and working hours pattern that are prevalent in Gulf, so as to lose minimum amount of time from the few days they get as holidays.

Apart from the preference, there is a related, but different aspect to timing. Suppose the indicated preference for a group of X passengers is 8 am. And we, unable to schedule a flight sharp at 8 am, schedule a flight at 9 am. What fraction of X will make use of this flight? We require an estimate of this **drop-off** factor with time along with the load-model in order to estimate the revenue for a given schedule. However, this also tends to be a difficult problem to solve. The drop-off may be determined by the arrival time at the destination, the type of passengers, as well as the nature of the next alternative. As yet another variation, suppose we schedule a flight at 7 am and another at 9 am, how will the X passengers be distributed among these alternatives: (a) take 7 am flight, (b) take the 9 am flight, and (c) take neither?

An ideal load model would list the number of passengers likely in a given sector, separated into various classes such as business traveler, families, personal travelers, etc. and for each of these classes we could associate a set of rules indicating their preferences when faced with a choice of flights, starting times, etc.

However, in reality this level of detail is not available. The classification of passengers recorded by the airlines is based on the charging scheme: first class, business class and economy class. Scheduling team has an intuitive understanding of the type of load across different sectors, which can be tapped as additional knowledge to get some feel for the information we require. For example, the staff may point out that preferred time for gulf flights to Indian destinations is Thursday night, since Friday is a holiday there and most of the passengers are those returning home for vacation. They also think in terms of parameters such as estimated percentage of market share that their own airline has on a given route.

4.2 Proposed Model

Keeping these considerations in mind, we have formulated the load model as a set of tuples of the form:

<start-location, destination-location, class-of-passengers, number of passengers, day-preference, time-preference>

Day-preference is assumed to be a flag indicating if weekend is preferred or weekday is preferred or there is no preference to the day of travel. Similarly time-preference indicates a preferred time or –1 to indicate that anytime is acceptable. The model can be revised easily to provide finer or coarser granularity for indicating preference. We expect the bulk of the data in the model to come from some statistical sources such as IATA data, suitably enhanced and moderated by human experts.

For load drop-off with time, we define a user-specifiable parameter called *load_drop_off_time*. The interpretation used is that no part of the indicated load will be lost, if the deviation of the actual flight from the preferred time is less than *load_drop_off_time*. Positive and negative deviations are considered alike. Note that this model of the load-drop is not integral to our system – it can be changed without affecting other parts of the system.

Given a time table and a load model, the revenue is estimated by first estimating the number of passengers in each flight, and then computing the revenue using the sector-revenue mapping. For estimating the load, the entries in the load model are examined one by one. For each load (for example, Mumbai-Frankfurt 200 passengers on weekend with no time preference), all applicable flight instances are identified from the schedule (all flights which covers Mumbai-Frankfurt and operating on weekend). These instances are then sorted based on some suitable priority scheme. A simple priority scheme may be increasing number of stopovers. More complex sorting functions can be

incorporated taking into account the type of passenger, etc. The available passengers are then loaded among these options subject to seat availability and other resource constraints. Passengers in one single load-model entry may be distributed among multiple flight instances (for example, 120 passengers on a Saturday flight and 80 on a Sunday flight).

This algorithm for estimating the load is only one possible approach. Given the granularity of load model, multiple estimates are possible using different algorithms. This is an inherent uncertainty in the domain. Taking into account the role of competing airlines and the unpredictable changes they make in their own schedules, the load estimate is bound to have a fair degree of variance.

Chapter 5. Airline Time Tabling Problem: An Analysis

In Chapters 1 and 3, we characterized the airline time tabling problem (ATP), specifying the general problem and the variety of constraints which comes into play. As we mentioned, the problem is of enormous practical importance. However, the typical ATP is heavily dependent on the airline concerned, government policies and so on. We need to formulate a problem based on the real-life scenario that can be academically studied and used by other researchers interested in this domain. In this chapter, we first formulate a simplified airline time tabling problem, named STP. This is described verbally and formulated mathematically.

As we saw in Chapter 2, scheduling is generally considered to be a hard problem. Many AI papers consider almost all cases of scheduling to be NP-complete in complexity. We study the STP analytically and prove that it is indeed NP-complete. We also justify that practical realizations of ATP are harder than STP. We believe this is an important result to formally understand the nature of this class of problems.

Another concern we have is to get a feel for the search space of this kind of problems. What is the cost distribution over a large set of solutions? How many (locally) optimal solutions can one expect to find? Are they related? And so on. We carry out an empirical study of the STP and report some results from this point of view.

Section 1 describes the STP and formulates a mathematical representation for it. Section 2 reports on the empirical study and the results obtained. Section 3 discusses the analytical study of the problem and Section 4 presents the proof of NP-completeness of STP. Section 5 presents some thoughts on the computational complexity of STP. Section 6 examines multi-variate optimization problems in general

and shows that it is NP-complete. While this is generally expected given the many problem domains of this class which have been shown to be NP-complete, we provide a formal statement of this along with associated proof.

5.1 STP: Simplified Airline Time Tabling Problem

The following assumptions describe the STP, which is a model problem. Just as the drosophila is a model insect for the biologists, the STP provides us with a simple and yet realistic model for studying airline time tabling problem within reasonable time and resource bounds.

1. There are 4 airports, named Alpha, Beta, Gamma and Delta.
2. There are 3 aircrafts of approximately similar nature. Two are of type AC250 and one of type AC200. AC250 carries a maximum of 250 passengers, and AC200 a maximum of 200. All the aircrafts are equal in all other respects.
3. The schedule period is restricted to one day with the flights operating from 4am to 10pm only.
4. The load-table is divided into 4 time points: 6am, 9am, 2pm and 6pm. Airport to airport demand is given for each time point for every pair of airports. We stipulate that the load drop-off is a sharp cut at 2 hours. That is, passengers desirous of traveling by 6am flight will not travel by a flight before 4am or after 8am, and do not mind any flight within this period.
5. The cost factors included are fuel charges per hour of flying, landing charges at each airport, and revenue per passenger for each sector. The revenue is assumed to be independent of the starting time of flight; it is a function of only the source-destination pair. Passenger-dependent charges should be deducted from revenue figures to make it more realistic.

6. All pairs of airports are considered as valid routes. These are enumerated and provided as routes.
7. There are no other constraints such as curfew time and landing time restrictions. Every aircraft need to stay for a minimum of 30 min at an airport from an arrival to the next departure. It may stay longer, as a part of the schedule.

The task is to generate a 4am to 10pm time table for each of the aircrafts given.

We denote the aircrafts (using "logical" names) as V_1 to V_3, airports as L_1 to L_4, time by an integer T denoting the number of minutes from 4 am ($0 \leq T < 1080$).

- LT(Li,Lj,T) is a function returning the number of passengers who can board a flight from Li to Lj at time T. Note that this may depend on time preference of the passengers concerned as well as the timing of any other flights covering sector Li-Lj around time T.
- RV(Li,Lj) is the revenue for a ticket from Li to Lj.
- FT(Li,Lj) is the flying time from Li to Lj. We assume that this is independent of aircraft type.
- CFT(Li,Lj) is the consolidated flying time from Li to Lj, taking into account the minimum ground time at Lj. Hence CFT(Li,Lj) = FT(Li,Lj)+30 in our model.
- FC, a constant, is the fuel charge per hour of flying. This also is assumed to be independent of aircraft type, in our current implementation; the dependency can be introduced easily, if required.
- LC(Li) is the landing charge at airport Li.

- PR(Li,Lj,Vk,Tm) is the profit[5] in operating a flight from Li to Lj using the aircraft Vk at time Tm. This is computed as

  PR (Li,Lj,Vk,Tm) = RV(Li,Lj) * LT(Li,Lj,Tm) –
  FT(Li,Lj)* FC – LC(Lj).

- The time table, TT, is a set of flights, where a flight F, is defined as a tuple <Li,Lj,Vk,Tm> as defined for PR above.

- For any two flights operated by a given aircraft Vk, F1 = <Li1,Lj1,Vk,Tm1> and F2 = <Li2,Lj2,Vk,Tm2>, assuming Tm1 <= Tm2 without loss of generality, the following must hold
  a) (Tm1+ CFT(Li1,Lj1) <= Tm2): *an aircraft cannot be committed to two things at one time.*
  b) if there is no other flight F3 = <_,_,Vk,Tm3> where Tm1 < Tm3 < Tm2, then Lj1 = Li2. *Flights must be continuous in airports, i.e., Beta-Alpha flight cannot directly follow a Beta-Gamma flight.*
  c) Tm1 >= 0 and Tm2+CFT(Li2,Lj2) < 1080 : *All flights operate within the 18 hour period from 4am to 10pm.*

- The objective function is now to maximize the net profit:

  Σ PR(Li,Lj,Vk,Tm) for all flights F = <Li,Lj,Vk,Tm> in TT

5.2 Empirical Study of STP

Search space of a scheduling problem describes how the profitability of a schedule varies as changes are incorporated in it. Given a point representing a particular schedule in the search space, we can visualize various types of changes that can be tried on that schedule. We can delete a flight, insert a new flight, change the aircraft of a flight, delay

---

[5] Note that the profit here is roughly the operating profit, and hence the calculation does not take into account fixed costs such as depreciation, lease charges, salaries, etc., and hence in absolute figures our profit values may appear to be high.

or advance a flight, etc. Each change results in a different schedule – some of which may be infeasible (such schedules can be represented with a profit of negative infinity). Some changes may increase the profitability and some may decrease it – the amount of increase/decrease depends on the specific change as well as the schedule in which the change is made. Adding a Delhi-Mumbai flight in an empty schedule and doing the same in a schedule which already has ten Delhi-Mumbai flights will bring in different amount of returns. The questions we wanted to address here are relating to the nature of the search space of the airline time tabling problem.

1. What does the distribution of profit versus number of solutions with that profit look like?
2. Are low-profit solutions more likely than good solutions?
3. How many very good solutions exist?
4. What is the general nature of the search space? Is it multi-modal or uni-modal? Is it vicious or benign?

A search space is vicious if it allows abrupt transitions in quality of solutions for small changes in the schedule. Formally, in a benign space one expects the difference in cost of two nearby schedules to be mostly proportional to the "distance" (we discuss the notion of distance later in this chapter) between the schedules. In a vicious space, this is not the case. Small changes such as addition/deletion of a flight can result in non-proportional changes in the cost of a schedule.

An exhaustive search of the space would provide clear answers to these questions. But that is too time consuming except for very trivial problems. Therefore, we resorted to a sampling approach. About 500,000 time tables were generated in a purely random manner and the profit frequency was plotted. The following were the typical graphs obtained.

The first one (see Figure 5.1) gives the actual number of solutions in various profit-brackets among the 500,000 solutions generated. The second graph (see Figure 5.2) shows the percentage of solutions with profit exceeding a given value (cumulative percentage of solutions).

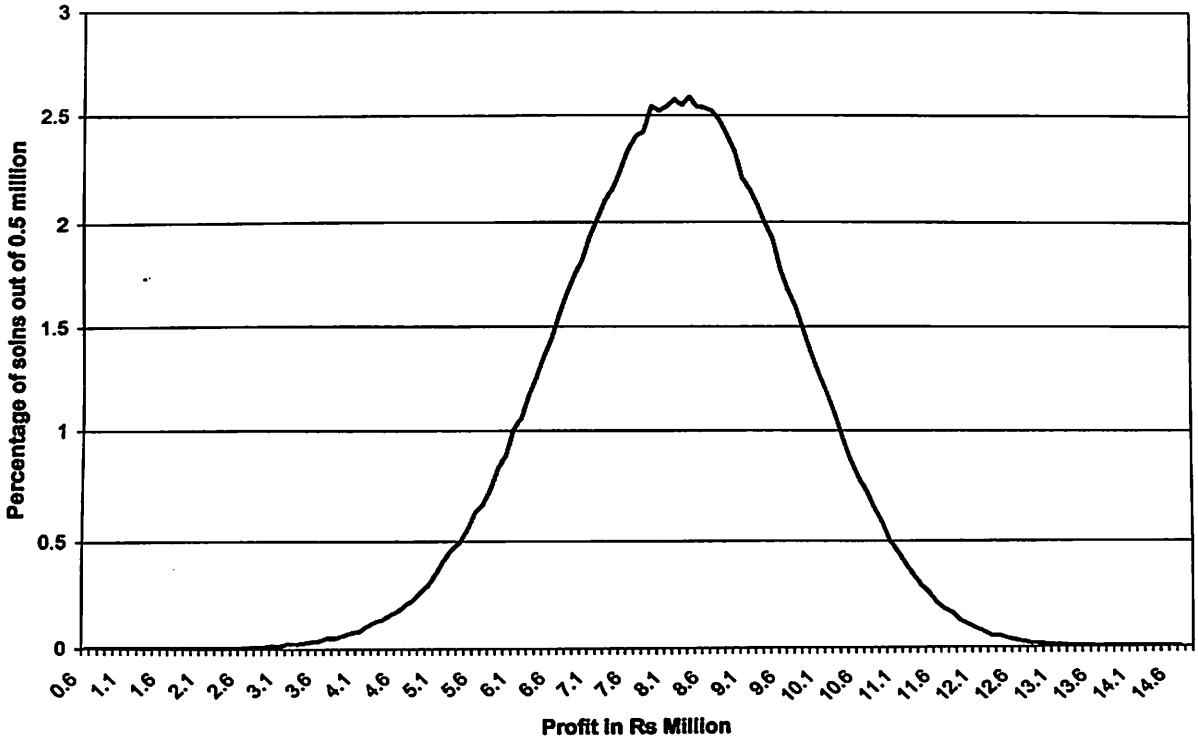


Figure 5.1 Profit distribution of solutions in a random sample of 0.5 million solutions.

Observations

1) The cost-frequency follows a near-Normal distribution with medium cost solutions being most likely and predominant, compared to very bad or very good solutions. Therefore, a purely random selection of solutions can be expected to give you only an average-quality solution, which can be quite poor compared to the best solution possible. Note that in the graph, the average solution earns only

about 50% of the profit[6] of what is the best reported as per this study. *The best here is what was found using only the random sampling. The actual best solution for the problem may be better than this.* Thus, the study to find better approaches to lead us closer to the actual best solution(s) is important.

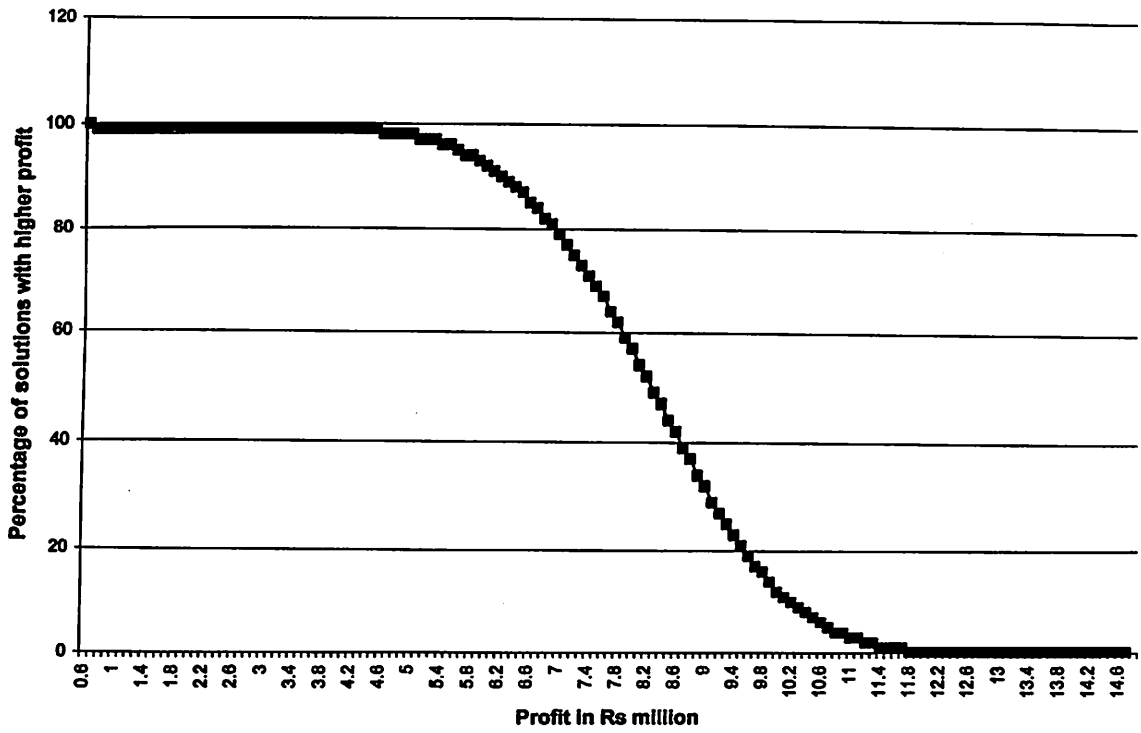


Figure 5.2 Percentage cumulative distribution of solutions against profit.

2) The variation in profit among the different possible time tables can be quite substantial. In this simple case itself, it ranges from 0.6 to 14.7 million Rs in profit.

3) The frequency graph was further analysed, in view of the observed similarity to Normal distribution. The graph was compared to the graph corresponding to Normal distribution with the same mean and

---

[6] Note that fixed costs are not counted in the profit calculation. Therefore, the ratio between profit of the best solution and the profit of the average solution will be much more, if fixed costs are also included.

variance as that of this data. Figure 5.3 plots the generated normal distribution curve as well as the observed curve.

The diamond-marked line is the original graph and the plain-line is the normal-distribution with mean=8.23 and standard deviation=1.54 (as obtained from the data). Except for the tapering-off effect at the extreme points, the similarity is quite high. In problems of this nature, it is not possible to get infinite profit or infinite loss – and hence our frequency curve will have a finite cutoff point at either side.

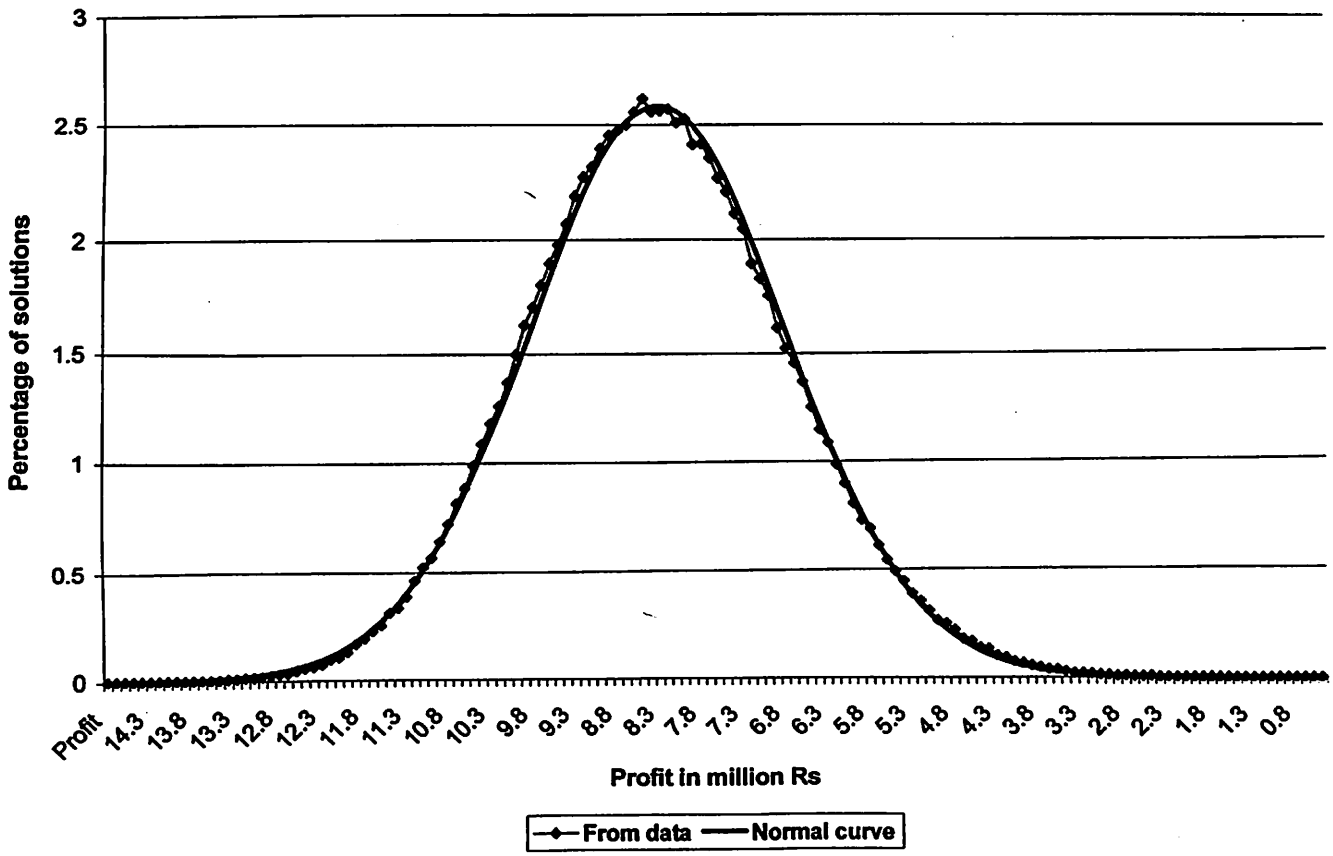


Figure 5.3 Distribution of solutions in the sample against profit and comparison against normal curve based on the same data.

Figure 5.4 shows the same distribution graph for a different dataset of the same class. The similarity of the curve to the curve observed from the earlier data set, goes to show that the behaviour is not unique to a particular dataset.

These observations play a useful role in explaining some of the behaviours in our later studies (see Chapter 7).

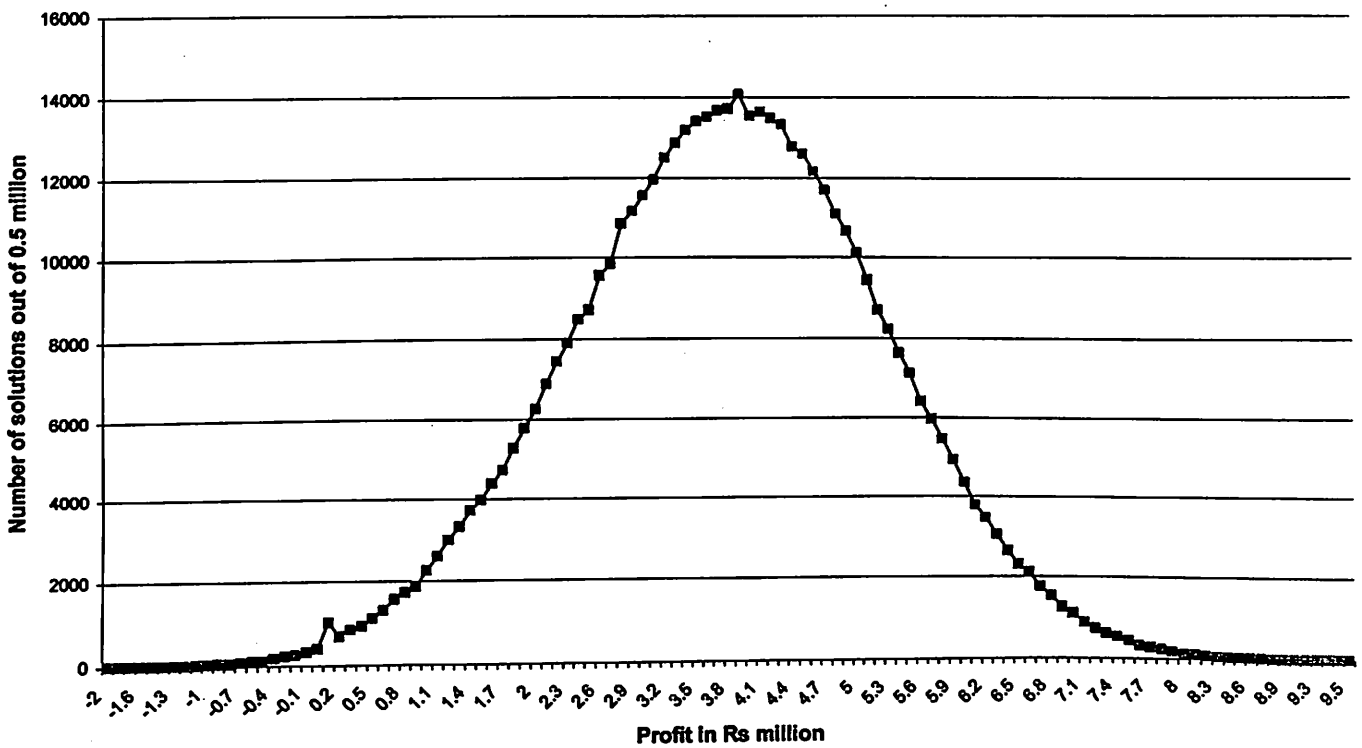


Figure 5.4 Profit distribution of random samples for dataset SIM1

5.3 *Nature of Search Space*

The complexity of search for the optimal solution in a search space is obviously dependent on the nature of the search space. In the extreme case, a uni-modal space as exemplified by functions like $f(x) = x^2$ is easy to search for optima. Even simple hill-climbing will not fail to reach the optima. When the space is multi-modal, with the different peaks having different profit values, the search need to be more sophisticated to ensure that it is not trapped in the smaller peaks. For real-life search problems, the search space will also be more complex containing plateaus, ridges and discontinuities, thus requiring more and more sophistication from the search algorithms.

In order to characterize the search space of any problem, we require a function to map a given solution to a point in the (typically) multi-dimensional space. This, in turn, requires a clear definition and

characterization of the various dimensions. A variety of models to define a search space is possible. For example, we could define the various pairs of airports as different dimensions and plot the number of flights on that sector as the coordinate along that dimension. Instead of the number of flights, we can use profit, total number of passengers carried, total revenue earned, etc as well. Instead of airport-pairs, we can use coarser granularity sectors (For example, India-Europe and India-Gulf), aircrafts, etc as the base for dimensions. Each choice gives a different characterization of the search space.

One primary consideration in defining a search space is the objective function (in our case, this will be the estimated profit). If two schedules give two different values to the objective function, they must map to two different points on the search space. In other words, given a point in the search space, it must correspond to a unique value for the objective function. Note that there may be more than one schedule corresponding to a given point; but they all must have the same value for the objective function.

Thus to characterize the search space for ATP, we require a mapping function from schedules to points in the search space, and thence to values of the chosen objective function. For a problem like airline time tabling, such mapping is very difficult to define, compared to other scheduling problems. For example, in school time tabling, one has a fixed-size matrix -- classrooms vs the periods --, which defines the schedule structure. Making each cell, representing the state of a given classroom in a given time slot, into a dimension gives you one fairly good mapping. The range of values can then be the set of various teacher-subject combinations.

In the case of ATP, neither the total number of flights nor the number of flights by each aircraft is known a priori. Thus, there are no fixed size structures to define dimensions. Therefore, one needs to look for suitable aggregation operators that can transform schedules to a form

with some regularity. Examples in the ATP domain are the total number of flights, number of flights by a specific aircraft, number of flights in a particular sector, number of flights in a given time interval, the specific timing of a flight, etc.

There is no universally good aggregation operator that satisfies the requirements of a search space characterization given earlier. For any of the operators listed above, it is possible to find two schedules which maps to a common point on the search space, but has different value for the objective function. Therefore, we need to make further assumptions about the profit function and the domain, in order to choose useful aggregation operators for analysis. This should allow us to treat certain differences as insignificant enabling clustering of schedules. For example, if the only difference between two STP schedules is that one has an Alpha-Beta flight at 9.00am, and the other has the same flight at 9.30am, the schedules are equivalent as per our STP definition.

If we choose to ignore passenger time preferences, counting the number of flights in various sectors (India-Gulf, India-US, etc) will give us a realistic sense of similarity. However, this is not realistic for our simplified ATP, for example, since the main concern here is the time of flight. Having 3 Beta-Delta flights, two in the morning and one in the evening is very different from having 3 such flights all during the noon time. Therefore, sector-wise aggregation is not a realistic yardstick for all ATPs.

Despite these concerns, we conducted some empirical studies of the search space. In order to study the nature of search space – benign or vicious – we examined some of the best solutions obtained in experimental runs, and compared them. There were two questions we asked:

a) Do the best schedules produced in multiple runs under identical configuration differ in cost? If so, what kind of distribution do we see?

b) When they produce solutions with same cost, are the schedules identical? If not, what is the nature of difference?

Note that we are using a stochastic model for schedule generation – so identical input conditions need not follow identical path resulting in identical schedule. Choices during the time table preparation process, where available, are made stochastically.

An analysis of the best schedules obtained over 10 different runs of the *full scheduling model* (dataset: ACT2) – not the simplified model - is shown below. Numbers along a column shows the sector-wise count of flights in the best schedule of one run. Columns are ordered left to right in decreasing order of total profit for ease of comparison. The first row gives a sequence number to the various runs (run 1 to run 9) and the second row shows the profit of the best solution reached during a run. The remaining rows show the number of flights in the specified sector in that solution. For example, the best solution of run 3 had a profit of 198.9 million Rupees (per week) and it had 2 flights in the sector Frankfurt-India and 39 flights in the Gulf-India sector. For this dataset, we have performed a sector-wise aggregation disregarding the time.

Run-number	**2**	**4**	**5**	**3**	**10**	**6**	**8**	**7**	**1**	**9**
Profit (in million Rs)	201.9	201.3	200.2	198.9	198.7	198.0	196.8	195.8	194.6	193.5
BangkokIndia	1	1	2	1	2	1	1	1	1	1
BangkokTokyo	2	2	1	1	1	1	1	1	1	2
FrankfurtIndia	2	3	3	2	2	2	3	3	5	2
GenevaRome	1	1	1	1	1	1	1	1	1	1

GulfGulf	6	5	6	6	6	7	7	7	5	6
GulfIndia	39	37	37	39	36	39	36	40	37	34
HongkongIndia	1	1	1	2	1	1	1	2	1	1
Hongkong-Osaka	1	1	2	2	2	1	2	2	2	1
IndiaBangkok	2	2	1	1	1	1	1	1	1	2
IndiaFrankfurt	2	3	3	2	3	2	2	3	3	2
IndiaGulf	39	37	37	39	36	39	36	40	37	34
indiaHongkong	1	1	2	2	2	1	2	2	2	1
IndiaIndia	25	32	30	27	32	27	31	30	28	24
IndiaLondon	7	7	7	6	6	7	7	6	6	5
IndiaParis	1	1	1	1	1	1	2	1	2	2
IndiaRome	1	1	1	1	1	1	1	1	1	1
IndiaSingapore	4	3	4	3	4	3	3	3	3	3
IndiaTokyo	1	1	1	1	1	1	1	1	2	1
LondonIndia	5	5	5	5	5	5	5	5	5	5
London-Newyork	4	5	4	4	3	5	4	3	3	3
Newyork-London	3	3	3	3	3	3	3	3	3	3
Osaka-Hongkong	1	1	1	2	1	1	1	2	1	1
ParisFrankfurt	0	1	0	0	0	1	1	0	2	1
ParisIndia	1	0	1	1	1	0	1	1	0	1
RomeGeneva	1	1	1	1	1	1	1	1	1	1

RomeIndia	1	1	1	1	1	1	1	1	1	1
SingaporeIndia	4	3	4	4	5	3	3	3	3	3
TokyoBangkok	1	1	2	1	2	1	1	1	1	1
TokyoIndia	1	1	0	1	0	1	1	1	1	1

The particular dataset used for this experiment, had no significant passenger time or date preferences in the demand table, and hence the choice of a schedule would essentially mean selecting good routes with adequate frequency. The table shows that there is no substantial difference or trend among the route-frequencies chosen among the 10 schedules. However, the profit estimated for these 10 schedules do differ by as much as 4% (minimum profit is 193.5 million per week and maximum is 201.9 million per week) – a significant variation[7], amounting to about Rs 416 million per year!

We also notice that the best solutions are all fairly close in terms of the distribution of flights. All solutions have identified the important sectors and approximate frequencies required for them. The differences are in further optimizations; which are understandably hard to get to during a stochastic exploration.

We also analysed a sample of the STP (dataset: SIM1) similarly. In this experiment also, we ignored the time difference and did sector-wise aggregration. The table obtained is shown below. The columns have been sorted in decreasing order of profit for easier comparison. The run-number gives a sequence number to the different runs of the program over this dataset.

[7] If we assume a fixed cost of 100 million Rupees, these numbers become 93.5 and 101.9 million Rs respectively. The 4% figure then changes to over 8%.

Run-number	**1**	**6**	**10**	**2**	**7**	**9**	**5**	**3**	**4**	**8**
Profit (in Rs million)	11.8	11.8	11.8	11.6	11.5	11.5	11.4	11.3	11.1	11.0
Alpha-Delta	2	2	2	3	1	2	2	2	2	3
Alpha-Gamma	1	1	1	0	0	0	0	1	2	1
Alpha-Beta	0	0	0	1	2	1	2	0	1	1
Delta-Alpha	2	2	2	1	2	1	1	2	2	2
Delta-Gamma	2	2	2	2	2	2	2	2	2	1
Delta-Beta	3	3	3	4	3	4	4	3	2	4
Gamma-Alpha	1	1	1	1	0	0	0	1	1	1
Gamma-Delta	2	2	2	2	2	2	2	2	2	1
Gamma-Beta	2	2	2	1	2	2	1	2	2	1
Beta-Alpha	0	0	0	2	1	2	2	0	2	2
Beta-Delta	4	4	4	3	5	4	4	4	2	3

Here, the maximum profit observed was 11.8 million Rupees per day. It was noticed that the best solution reported in many runs had this profit. Note that in the earlier case, identical cost was rarely reported in more than one run.

A few solutions with identical profit were examined manually for similarity. They were found to be identical in some cases, and equivalent sector-wise in other cases. Thus the profit equivalence was not incidental – but arose due to the same demands being serviced through similar flights.

The sector-wise analysis here does not show any noticeable trends. Unlike in the earlier case, the best solutions obtained in different runs do not share any major common set of flights. Note, in particular, that sharing the same sector-wise distribution does not imply a good solution; an example is run number 3. It has the same sector signature as the best solution, but has a poorer profit figure. Timing of the flights makes a significant difference in this dataset.

Considering these observations we hypothesise that:

a) The nature of the solution-space is likely to depend heavily on the constraints in force. If the passenger preferences are significantly time dependent, the solution-space is likely to contain many local maxima spread out in the search space. Otherwise, high-quality solutions seem to be comparatively concentrated in a region. Within this region, there are local optima – but the solutions share a major part of the solution structure.

b) Using single perturbation operators, it is difficult to escape from local optima towards the end of the run. Examining the solutions corresponding to the various local maxima shows that a cluster of perturbations would be required to transform one of these solutions to another. Making one perturbation at a time leads to poorer solutions at intermediate stages and hence the intermediate solutions are rejected without being given a chance to explore further perturbations. This is one issue of concern in using perturbation models.

Subsequent experiments (reported in Chapter 8) concur with these observations. The search space is not totally vicious – abrupt and large magnitude variations are not seen mostly. An analysis of changes in profit caused by a single application of the perturbation operator showed that more than 80% of the time, the resulting change in profit was less than 3%. This was true irrespective of the profit value of the current solution. As the profit of the source solution increases, more and more of the perturbations lead to a decrease in the profit value; the absolute change satisfying the observation above. In a highly vicious space, a much sharper change in profit would be expected.

The number of different local maxima, very likely, depends on the problem instance. Tighter passenger time preferences, abnormal revenue or cost variation on particular sectors (a particular route has a disproportionately large fare, for example), etc are likely to be significant factors determining the shape of the search space. They contribute to increasing the number of local maxima and the relative difference between the various local maxima.

5.4 Complexity of STP

For complexity analysis, particularly checking for NP-completeness, it is necessary to cast the problem as a decision problem. The normal practice [Baase, 1988] is to use a threshold number and cast the problem as *is there a time table with profit greater than K?*. We assume that the problem is stated in this way for this section.

5.4.1 What is NP-completeness?

Based on the run-time complexity, algorithms can be classified as polynomial (where the worst-case complexity is a polynomial function of the problem parameters) and non-polynomial (all other cases). Problems with a polynomial complexity algorithm, are said to belong to P-class. Among the non-polynomial, we have the unsolvable problems such as the halting problem, and a great many problems with

exponential complexity ranging from a^n to n^n, where a is a constant and n is a measure of the input size.

Among these is a class called NP; the abbreviation stands for "non-deterministic polynomial". Problems in this class are cast as a yes-no decision type. For example, does the given graph have a Hamiltonian path? Is there an assignment of truth values to the variables in a Boolean expression in conjunctive normal form (CNF) that makes the expression true? And so on.

Such problems belong to NP if a given candidate can be verified to be a solution (or not) in polynomial time. For example, given an assignment of truth values, test whether the CNF expression evaluates to true. The solution to these problems can be cast as a series of decisions. For example, pick true/false as the value for each variable in turn. In the Hamiltonian path problem, choose the next vertex to visit. Viewing the solution this way, the NP requirement can be explained also as follows: If an oracle were available to make the right choice at each decision point (note the non-determinism here), the solution can be found in polynomial time. This explains the name NP. NP is a superset of P.

Identifying the complexity class of problems is a rich field and of considerable practical interest. P-class problems are amenable to solving in real-time for fairly large problem sizes, where as exponential class problems suggest looking for heuristic or approximation models. In the NP sphere, there is an interesting class of problems, called NP-complete problems. The interest stems from the following observation about them.

1. The problems are all of high practical significance and hence finding good algorithms is of high importance.
2. They span a variety of domains from logic and graph theory to resource scheduling.

3. No polynomial time algorithms are known for any of these problems; but there is no lower bound proved so far, to eliminate polynomial solution.

4. These problems are interlinked by polynomial time reductions, so that if any problem in the set is found to have a polynomial time solution, all problems can be solved in polynomial time.

What is a polynomial-time reduction? Reducing a problem X to another problem Y (e.g., Boolean satisfiability to knapsack problem) involves the following steps.

1. devising a transformation, t1, of any valid input to X to some valid input to Y,

2. devising a transformation, t2, of valid outputs of Y to valid outputs of X, possibly under assumptions permitted by the requirements of X (For example, boolean satisfiability enables you to assume that all the variables are boolean valued), and

3. proving that the sequence, transformation of the given input by t1, solving Y using the transformed inputs and transforming the resulting output by t2, produces the same output as solving X using the original input.

If the two transformations, t1 and t2, can be done in polynomial time in the amount of input to X, then the reduction is said to be polynomial time reduction. If there is a polynomial time reduction from X to Y, and if it is known that X has no polynomial time solution, then we can conclude that Y cannot have a polynomial time solution. Otherwise, transformation t1, followed by solving the resulting Y problem followed by the reverse transformation t2, would amount to a polynomial time solution for X. This is the basic idea behind proofs of NP-completeness.

As of now, no one has been able to find a polynomial time solution to any of the problems in this set, nor has any one ruled out a polynomial solution, leaving this as, perhaps, the biggest open problem in

computer science. It is generally believed, however, that NP complete problems do not have polynomial solutions. Thus if a problem is proved to be NP-complete, it is considered to be computationally intractable.

To show that a problem X is NP-complete, we need to show that it is in NP and that for some problem Y which is known to be NP-complete, there is a polynomial time reduction which transforms Y to X [Baase, 1988].

5.4.2 ATP is in NP

First of all, let us check if ATP belongs to the class NP of complexity. Given the formulation in Section 1 of this chapter, we can see that verification of whether the net profit in a time table exceeds K or not, requires a one-time loop through all the flights, summing up the PR values for each flight and at the end checking if it exceeds K. This is of linear complexity in the size of the time table. Therefore, the ATP problem, being verifiable in polynomial time, belongs to the class NP.

It is also easy to see that there is a straight-forward algorithm for devising the time table, which is of exponential complexity. A broad outline of this algorithm would be as follows:

Main()

```
Set profit = - infinity; Set bestTT = null;
Set TT to be empty
expand(TT)
```

Expand(CurTT)

```
For each vehicle Vk
   Li = last_location of Vk according to CurTT
   For each applicable route R from Li
      Construct a flight from Li via R, and add to CurTT;
      If no new flight is possible, output CurTT and return;
      Let newTT be the resulting state.
      Evaluate newTT. If newTT is better than what has
          been found so far, revise profit and bestTT
      Expand(newTT)
Return;
```

If F is the maximum number of flights that an aircraft can operate during the time interval of interest, the depth of recursion is bounded by |V|*F. At each level, there are |V|*|R| possibilities (options are to select a vehicle and select a route). If all vehicles are identical, we can ignore the vehicle selection dimension, and force a specific order for selecting vehicles. First expand V1 to maximum, and then V2 and so on. But, in general, the different aircrafts are not identical; as mentioned earlier, there are a number of parameters associated with different types of aircraft. Therefore, to be safe, we must try all possible orderings of vehicles (v1-v2-v3, v1-v3-v2, v2-v1-v3, etc) and hence choice of vehicle has to be retained as a branch point in the search.

The number of routes possible at any point in the algorithm is normally much less than the total number of routes available (|R|), since the routes are normally severely restricted by the current location. If delays are allowed in the schedule (For example, wait for an hour before the next flight), then that also adds to the possibilities at a level. Thus we get the number of states to be explored as of the order of $|R|^{|V|*F}$. A typical figure for this, based on the simplified problem, comes to 9^{40} –

obviously too large a number for exhaustive enumeration. For large airlines with hundreds of aircrafts and routes, the search space is mind-bogglingly large.

5.4.3 Knapsack Problem is Polynomial Reducible to ATP

We will now try to show that a known NP-complete problem is polynomial reducible to a simplified version of ATP. This simplified version is not the STP mentioned earlier in this chapter, but another simplification as explained below.

We will take the knapsack problem -- currently known to be an NP-complete problem -- as the candidate problem here. Suppose we have a knapsack of capacity C (integer > 0) and N objects with sizes S_1 to S_N and profits P_1 to P_N. S_I and P_I are integers > 0. The decision problem asks: *Given a number k, is there a subset of the objects that fits into the knapsack and has a total profit of at least k?* We need to define an ATP using these parameters, so that the solution of that ATP can be transformed to obtain a solution to this problem.

Consider a simpler version of the ATP with a few additional constraints. The number of airports is N+1 and are labeled 0 to N. Airport numbered 0 is the base airport. The only valid routes are <0,I,0> for all I: 1..N. That is, aircraft leaves base 0, visits one other airport I, and returns to base immediately. We also assume that there is only one aircraft.

Let S_I be the time taken for a flight to airport I (including return), and P_I the profit estimated to be made by the flight. C is the schedule period, the time within which all flights have to operate. We are also assuming that there are no time constraints for the passengers; the passengers are available in equal numbers irrespective of the time of the flight. Curfew timings etc are also not applicable. And we ask the question: *is there a combination of flights, which can be completed in time C and generates a profit of at least k?*

Note that the transformation involved is straight-forward. The inputs to be produced for this restricted ATP are the routes, the profits and travel-time. Each of these takes only O(N) time to generate because each airport gives rise to one and only one route, profit and travel-time entry. Note that one of the end points of all legs is the base location. Thus the problem transformation is of polynomial complexity in time.

Let the knap-sack problem above have a solution L representing the set of objects picked. Then,

$L = \{r_1, r_2, \ldots, r_k\}$ $k <= N$, r_i: 1..N. *eg: {1,4,6,7,8}*

$\Sigma_{(i \text{ in } L)} S_i <= C$, $\Sigma_{(i \text{ in } L)} P_i >= k$

This is also a solution for the simplified ATP mentioned above. r_i (for i in 1..k) indicates a flight from location numbered zero (that is, the base) to location numbered r_i and return to base. The time taken by the k flights is $\Sigma_{(i \text{ in } L)} S_i$; this is at most C. The total profit is $\Sigma_{(i \text{ in } L)} P_i$, which is at least k. Therefore, if there is a solution to the knap-sack problem as per its original formulation, then the transformed ATP problem also has a corresponding solution. The same reasoning can be used in reverse to show that if the transformed ATP has a solution, there is a uniquely defined corresponding solution for the knapsack problem.

Therefore, the knapsack problem is no harder than the ATP problem (if it was, we could use the transformation to ATP and solve it as an ATP). Since the knap-sack is an NP-complete problem, this classifies ATP also as an NP-complete problem.

5.5 Computational Complexity

What is the computational complexity of solving ATP, specifically using a perturbation model? Is it possible to estimate some bound on the number of iterations or run-time?

As mentioned earlier, most problems of this nature have been proved to be of exponential complexity in the worst case and have been shown

to be NP-complete. In the last section, we proved ATP also to be NP-complete. However, worst case is not necessarily a useful guide in practice. Thus the question is what kind of complexity does one expect on practical problems?

The complexity of our perturbation algorithm can be defined as the complexity of one iteration * number of iterations. One iteration consists of evaluating the individuals currently present in the population, selecting n of them ($\leq$ N, the population size) for perturbation, generating the perturbed solutions, and discarding a fixed fraction of the resultant set of solutions. Complexity of one iteration is estimated as follows.

Evaluating an individual: (a) Estimate the load by checking which entries from the table can be met by different flights in the schedule. This is O(L*|V|*F) where L is the number of entries in the load table, |V| is the number of aircrafts and F is the maximum number of flights that can be operated by an aircraft during the schedule period. (b) Evaluate the profit of the schedule by computing the revenue and cost of each flight = O(|V|*F).

Generating new solutions: Generation of a solution is by perturbing one or more chosen time tables. Details of the perturbation operators are discussed in Chapter 7. The primary operations are deletion of a flight and insertion of a flight. Deletion requires choosing a flight from the current time table. If a random selection is used, the complexity of this process would be O(1); if a roulette wheel is used, the complexity will be O(|V|*F). Insertion also requires identifying the point of insertion and then choosing a route for the new flight. Insertion points are normally randomly selected or identified by the preceding deletion operator. Thus the insertion complexity would be O(|R|); but in practice, the number of useful routes at any point will be less than O(|V|*F).

Thus, total complexity of generating n new solutions = O(n*|V|*F). The set of solutions need to be sorted. If the existing solutions are always kept sorted, the incremental sorting can often be done by simple insertion sort as and when new solutions are generated. Thus the cost of sorting is negligible compared to the cost of generating a new solution.

Thus the overall complexity of one generation is O(n*L*|V|*F)+O(n*|V|*F) = O(L*n*V*F). The size of the load table is determined largely by the number of locations. "n" is normally a fraction of the population size N. Thus, the iteration time is proportional to a polynomial function of the number of locations, number of aircrafts and the population size. The run-times we have observed during the studies reported later in the thesis, show a near linear growth with population size, as implied by the above estimate.

The difficult factor in the complexity estimate is the number of iterations. This is often a matter of subjective judgement. Given the stochastic nature of the problem, this cannot be a definite number. Often, we consider the point of convergence of the solutions in a population as determining the number of iterations required. As will be seen in the experiments later on in Chapter 8, this number also varies significantly from run to run.

Some empirical research has shown that the effort required to find a solution for classic problems such as graph colouring shows a distinct pattern of easy-hard-easy. For example, measuring the complexity of a graph in terms of its connection density (number of edges/number of vertices), the following behaviour has been observed [Cheeseman et al, 1991]. For low values of this parameter, it is easy to find a satisfiable colouring. As the parameter increases, the difficulty of the task increases, showing a sharp increase within a small value-range of the connectivity parameter. The difficulty then reaches a peak and starts dropping in a similar manner. For large values of connectivity, the

Thus, total complexity of generating n new solutions = O(n*|V|*F). The set of solutions need to be sorted. If the existing solutions are always kept sorted, the incremental sorting can often be done by simple insertion sort as and when new solutions are generated. Thus the cost of sorting is negligible compared to the cost of generating a new solution.

Thus the overall complexity of one generation is O(n*L*|V|*F)+O(n*|V|*F) = O(L*n*V*F). The size of the load table is determined largely by the number of locations. "n" is normally a fraction of the population size N. Thus, the iteration time is proportional to a polynomial function of the number of locations, number of aircrafts and the population size. The run-times we have observed during the studies reported later in the thesis, show a near linear growth with population size, as implied by the above estimate.

The difficult factor in the complexity estimate is the number of iterations. This is often a matter of subjective judgement. Given the stochastic nature of the problem, this cannot be a definite number. Often, we consider the point of convergence of the solutions in a population as determining the number of iterations required. As will be seen in the experiments later on in Chapter 8, this number also varies significantly from run to run.

Some empirical research has shown that the effort required to find a solution for classic problems such as graph colouring shows a distinct pattern of easy-hard-easy. For example, measuring the complexity of a graph in terms of its connection density (number of edges/number of vertices), the following behaviour has been observed [Cheeseman et al, 1991]. For low values of this parameter, it is easy to find a satisfiable colouring. As the parameter increases, the difficulty of the task increases, showing a sharp increase within a small value-range of the connectivity parameter. The difficulty then reaches a peak and starts dropping in a similar manner. For large values of connectivity, the

graph is impossible to colour and failure can be reported easily, for example, by finding a vertex with more adjacent vertices than the number of available colours.

Similar "phase transition" behaviour has been found for a number of other exponential complexity problems such as traveling sales person, constraint satisfaction and satisfiability [Mammen and Hogg, 1997]. It is possible that bigger problems such as scheduling may also show such behaviour – however, hardly any results are so far known. One major issue in such an approach is to identify the base parameter, which determines the complexity.

Given that measures such as worst-case complexity are not reliable indicators of complexity on practically significant problems, and the infeasibility of obtaining information such as average-case complexity, attention has been turned to empirical analysis of algorithms. [Hooker, 1994] advocates this approach vehemently as the only practical approach to understand these algorithms. Stochastic and heuristic algorithms make the traditional analysis of complexity irrelevant, giving greater credibility to Hooker's claim for empirical studies of algorithms.

5.6 The Multivariate Optimisation Problem is NP-complete

In this section, we prove that Multivariate Optimisation (MVO) Problem, in general, is NP-complete. We formulate MVO as follows:

Is there a point, P in a given multivariate space, which offers a value for an evaluation function, g, that is higher than a specified threshold. In other words, the problem is to say if there is a P such that g(P) > threshold.

Consider the space of all points defined by a set of n variables (V_1, V_2, V_3, ..., V_n), where each of the variables is constrained to take either 0 or 1 as its value. And consider a function in the form:

$$f(P) = C_1 * C_2 * C_3 \ldots * C_k$$

Where each C_i is a clause, that is a well formed expression combining a subset of 2n literals[8] defined over the n variables, using the "+" operator signifying Boolean addition. Here we use a Boolean function f as the basis for computing the evaluation function g(p). Let there be a numerical value associated with the literals l1, l2, l3, etc. This value n(li) is "1" iff li is true, and "0" otherwise. With each clause Ci associate a numerical value n(Ci). This value is the sum of the numerical values associated with each of the literals in that clause Ci. Thus, g(p) = n(C1) * n(C2) * ... * n(Ck), where "*" denotes arithmetic multiplication. Then, we have the following relation between f(P) and g(P).

g(P) is 0 if and only if f(P) evaluates to FALSE.

The MVO, in this version, is to answer the following question:

Is there a point in the given space which gives a value to g(P) that is greater than 0?

Now, let us consider the clause satisfiability problem from complexity theory. This problem is to find an assignment to the n Boolean variables involved in a given set of k clauses, such that every one of the clauses is true for that assignment. We note two points:

1) The MVO problem defined above is in NP, because given a point p, we can compute in polynomial time if g(p) is "1" or not.
2) The MVO is NP-complete, because the CSAT problem is NP-complete, and any given CSAT problem in n variables can be mapped onto an MVO problem, by using the product of the numerical values of its clauses as the evaluation function. Any point that constitutes an answer to the MVO problem will automatically be the answer to the CSAT problem as well.

---

[8] A literal is either a Boolean variable, or its complement.

The evaluation function defined above creates a truly vicious space, in which the evaluation function can report totally independent values for a point and its neighbour (i.e. another point which differs from the first one only in the value of one of the n variables). The classic proof of the CSAT problem being NP-complete is, in fact, based on this independence.

We note that the evaluation function defined above is the product of numerical values of its clauses. It is this multiplicative product operator, which introduces the vicious property of the space here. On the other hand, consider other evaluation functions that involve no product of clauses. Consider only those functions which use arithmetic + and – operators, and no multiplication or division operators. Then the value of the evaluation function g() for an immediate neighbor of P would be related to g(P).

Chapter 6. Implementing the Perturbation Model

In this chapter, we discuss the perturbation model implemented for the problem as discussed in chapter 3 (in other words, we are not restricting to the STP).

6.1 Representing a Time table

Please recall the time table representation outlined in Chapter 1. We use that model to define the types of perturbations. Pictorially, the model is shown in Figure 6.1.

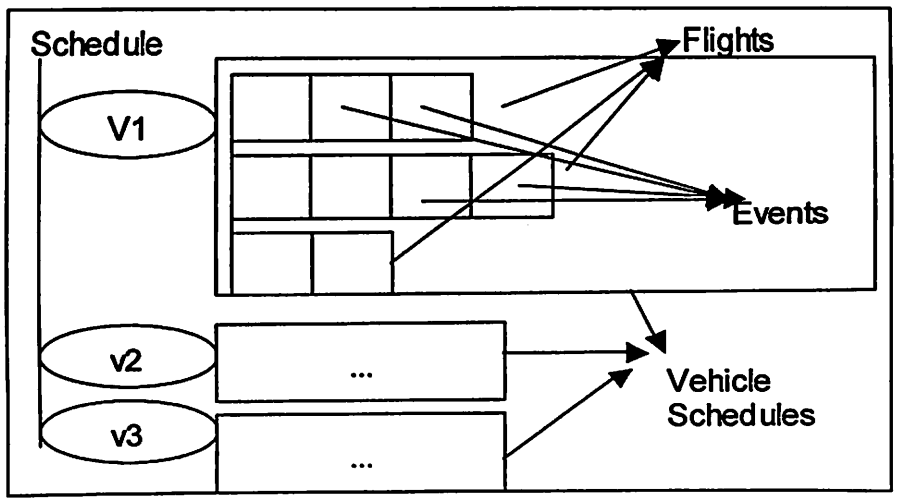


Figure 6.1 Pictorial representation of a time table

The outer structure is called a "Schedule" which represents a complete time table. It, in turn, has a list of aircrafts and a "vehicleschedule" for each aircraft. Each vehicleschedule consists of a sequence of flights. A flight has an associated route, starting from departure from the first location in the route to arrival at the last location of the route. Each leg of flight is called an event.

An event, thus, consists of:

- the source location
- the destination location
- departure time from the source location
- arrival time at the destination location
- load in the aircraft at the time of arrival at source
- any loading from the source location
- any unloading at the destination

For example, an Alpha-Beta-Delta flight may be represented as two events as follows:

Event1: [vehicle:ai300-1; sourcel:Alpha; destl:Beta; routeid:r1; Ddate:1pm; Adate: 2pm; arrival_load: nil; loading: ((economy Alpha Beta 100) (economy Alpha Delta 100)); unloading: ((economy Alpha Beta 100))]

Aircraft labeled ai300-1 leaves Alpha at 1.00pm carrying 100 economy class passengers destined to Beta and another 100 to Delta, reaches Beta at 2.00pm, unloads the 100 passengers destined to Beta, ...

Event2: [vehicle:ai300-1; sourcel:Beta; destl:Delta; routeid:r1; Ddate:2.30pm; Adate: 4.30pm; arrival_load: ((economy Alpha Delta 100)); loading: ((economy Beta Delta 100)); unloading: ((economy Beta Delta 100) (economy Alpha Delta 100))]

... leaves Beta at 2.30pm loading 100 passengers destined to Delta, reaches Delta at 4.30pm and unloads the 200 passengers on board destined to Delta.

6.2 Datafiles

The various inputs required and the way they are specified to the system are as follows.

a) File: **config**

Specifies a number of parameters to configure the system for a particular run of the system. The parameters include

i. datadir (the directory containing the actual data files)

ii. population size

iii. initfile (whether the population is to be initialized randomly or read from a file)

iv. niter (number of generations for which the program should be run)

v. persistence factor (explained later)

vi. crossover (what fraction of the new solutions are to be generated by crossover; the balance will be by mutation)

vii. repeat_from, n_repeat: used for running the program multiple times. Specifies the program to run with runid varying from repeat_from to n_repeat.

viii. rm_duplicate: duplicate solutions to be retained or not.

ix. del_stop, ins_prob: probabilities controlling the behaviour of the mutation operator (explained later).

x. strategy (retention strategy to be used)

xi. record_file (name of file in which to record log of the run, used for collecting data for our studies)

xii. Npop: the number of islands to be used (see Chapter 9).

b) File: **routes.in**

Contains the routes used. Example: `2 r1 Beta Delta`. This specifies a route named r1 consisting of two airports, which is a point-to-point connection from Beta to Delta.

c) File: **revenue.in**

Specifies the passenger revenue, for each source-destination pair and type of passenger. Eg: Alpha Delta eco_pass 5700.

d) File: **event_charge.ini**

Specifies the various charges applicable to an event (See Chapter 3 for the types of charges and their representation.). Eg: Beta * * * * 20000 landing. Indicating that 20000 Rs are payable as landing charges for each flight landing at airport Beta.

e) File: **item_store.ini**

Load-table as discussed in Chapter 4. Eg: Alpha Delta eco_pass -1 840 100. About 100 passengers are estimated to want to fly from Alpha to Delta at 840 minutes[9] time in economy class on any day.

f) File: **location.ini**

List of locations and their relevant attributes. Eg: Beta Bt 0 1 *. The fields are the full name, abbreviated name for use in compact displays, GMT-offset, base flag, and types of aircrafts allowed at that airport. The GMT-offset is used to produce display in local time when flights spanning time zones are involved, and to map airport specific constraints (often specified in local time) to the internal time scale. Base flag indicates if the airport is a base location – the primary significance is the presence of maintenance personnel at such locations. If a location is base, aircrafts spend a little longer there between landing and takeoff, compared to other locations. The example lists a location Beta, whose short name is Bt, time offset is zero, is a base location and allows all aircrafts to land/take off.

---

[9] Data files and internal data structures represent time as number of minutes elapsed from the start of the time table.

g) File: **item.ini**

List of passenger types.

h) File: **link_charge.ini**

Charges on a link basis (from-to pair). Eg: *** bo747 time 94000 fuel. An aircraft of type bo747 spends Rs 94,000 per hour as fuel charges for all flights. Section 3.4 explains the rationale of this notation.

i) File: **masterdata.ini**

This specifies general data for an airline. The data include types of aircrafts (details to be given in vehicle.ini), minimum turn around time at an airport, schedule period and load_drop_off_time.

j) File: **vehicle.ini**

Information on each aircraft available including type, range, speed, seating capacity, etc. Eg: ab310:1 ab310 200 854 7520 480 45000. There should be one entry for each individual aircraft available for time tabling.

The system is invoked with the configuration file as a command line parameter. The other data files are to be located in the datadir specified in the config file, and will be loaded automatically at the start of the run.

A sample config file is shown below:

```
p_factor        0.5
datadir         ../SIM1
record_file     dsp2c
populationsize      100
strategy        three
initfile        random
del_stop        0.3
ins_prob        0.2
```

```
crossover   0.4
rm_duplicate        no
niter       2000
n_repeat    20
repeat_from         0
Npop        1
```

The system generates a log file noting the major events in the run for analysis and verification. It also produces a statistics log file for each run (specialized using the runid, and the logfile option in the config file) recording the cost of the best solution at each iteration. Most of the analysis we have done is based on this output.

At the end of each run, it saves all the schedules in the current population into separate files.

6.3 Perturbation Model

The perturbation model for solving a problem consists of 5 main components:

1) Initialization of the population
2) Evaluation of an individual solution
3) Perturbation operators
4) Population management
5) Termination mechanism

These are now described in detail.

Initialisation: We use a multi-seed population for perturbation. This is the primary protection mechanism to reduce the probability of getting stuck in local optima. This also allows us to combine partial solutions from multiple individuals, borrowing elements from multiple solutions. The initial population of N elements is initialized randomly. The number N is specified in the configuration file mentioned in the last section. N

Evaluating a Time table: Evaluating the schedule is estimating the amount of profit that the time table will bring when deployed. Deploying a schedule here means applying the demand model we have for the problem and estimating the number of passengers in each leg of the various flights in the time table. Then we compute the revenue based on this estimated loading, and subtract the cost. We get the profit. The higher the profit, the better the schedule.

The demand tuples in the load-model are examined and for each demand tuple, all flights that can cater to that demand are identified from the schedule. The flights can be ordered based on any domain specific preferences if required. In our case, since we are using a simple load-cut-off model, no sorting is done. The passengers in the demand tuple are assigned to the selected flight subject to capacity constraints. If there is more load remaining, the next flight is selected from the list.

A demand tuple <*sourceloc, destinationloc, type, count, pref-day, pref-time*> is applicable to a flight if

1. in the flight route, *sourceloc* and *destinationloc* are no more than 1 stop-away from each other
2. the preferred day *pref-day*, if specified in the demand, matches the day of flight
3. the preferred time *pref-time*, if specified in the demand, is within Dt units of the actual departure time from flight. Dt, the load_drop_off_time is specified in the master data file.

The number of passengers that can be accommodated in the selected flight is the minimum of the spare capacity for *type* class in the flight from *sourceloc* to *destinationloc*, and *count*. The current demand-table entry is then updated, removing the passengers loaded into flights thus selected. This process is repeated for all demand table entries. Any

demand table entries remaining at the end of this process are those that cannot be catered to by the current schedule.

The revenue and cost is then computed by scanning the event-charge and link-charge files and computing all applicable costs taking into account the number of passengers, flying time, etc.

The computation can be enhanced by adding pseudo costs, for constraints whose violation cannot be represented directly as Rupee value.

Perturbation Operators: Perturbation operators are, perhaps, the most critical component of a perturbation-based approach. The function of these operators is to transform existing solutions through well-defined steps to different solutions, normally making revisions to some selected parts of the solution. In general, a wide range of such transformations is possible. For example, in our case, the transformations can range from delaying or advancing the departure time of a flight to adding/deleting one or more flights. As discussed in Chapter 2, a number of issues have to be kept in mind, while choosing perturbation operators.

The operators are of two types: crossover operators and mutation operators (following GA terminology). Crossover combines features from two or more existing schedules and creates one or more new schedules. Mutation makes some modification on a single schedule. We implemented different types of operators. These are discussed, along with the rationale for selecting them, in the next Chapter. Issues such as relative frequency of applying the different types of operators are also discussed.

Population management: The third aspect to be handled is management of population across generations. Population management covers a variety of issues including selection of schedules for perturbation, selection of schedules to be discarded/retained from

iteration to iteration and ensuring mechanisms for creating/maintaining diversity. Considering the various issues mentioned earlier, we implemented a few different strategies for selecting the individuals to be retained from one generation to the next. As a default, we use the "retain best-half" strategy (named s1). In this, after generating additional N schedules from the existing population of size N, the resulting population is sorted in descending order of profit, and the best N are retained. Other strategies and their rationale are explained in the next Chapter. Other issues in population management and our approach to them are also discussed in Chapter 7.

Termination: Termination of the program is currently done based on the number of iterations. This is primarily to help experimentation. In practical contexts, we may use termination based on no-progress for a number of iterations or reaching some particular threshold for the value of evaluation function (in our case, the profit).

6.4 The Implementation

The system was implemented using Java [Horstmann, 200]. It can run on Windows or Unix platforms. There is no significant graphical UI component, making the system portable across platforms.

In order to support distributed programming ideas (subject of Chapter 9), the system has been implemented using a master-slave architecture. The master and slave components can run on any machine independently. In the sequential model, the master primarily keeps track of the various active slaves and displays their status (current generation number, best profit so far, generation number when the profit improved last, etc). It has more functionality in the context of the distributed system model described in Chapter 9.

The slave is essentially responsible for running the ATP model. On invocation, it establishes connection with the specified master, initializes the population as per the configuration file directives and runs

the system one iteration at a time till the specified iteration count is over or the master instructs it to stop the execution. Each slave maintains its own log file and a record of the best solution found so far.

The major classes in the implementation, and their functionality are described in Appendix A. Overall about 4500 lines of Java code has been implemented. Over a thousand configurations have been run spanning 4 – 5 data sets.

6.5 Other Relevant Issues

We now examine some domain specific issues that an implementation for ATP has to address.

6.5.1 Where Are the Aircrafts Initially?

For a problem of this type, there is no time-point in the schedule where all aircrafts are available at a single location. Therefore, it is not possible to view this as a single depot vehicle-scheduling problem [Bodin et al, 1983]. Indeed, it is a multi-depot problem with the additional complication of not knowing how many aircrafts are available at each depot. This makes it difficult to start with a known initial position in generating schedules.

We solve this problem by assuming a location named "dummy" at which all aircrafts are positioned at time 0. The location dummy is defined in the model in such a way that there is zero distance from dummy to any location and hence any aircraft can reach wherever it is required without loss of time. This also makes it easy to handle deletion of flights by not having to handle empty schedules for any aircraft as a special case. Since the dummy event need not (and should not) be deleted, that event will always be there in any valid schedule.

6.5.2 Slot and Curfew Timings

A second problem is handling of slot and curfew timings. These imply that some intentional delays may be introduced in the time table for

some aircrafts, so that they do not land during curfew and non-slot timings at any airport. In order to simplify the studies, we have ignored the possibility of parking aircrafts at airports except at start of any flight. Thus if a scheduled flight is likely to land at an airport during any non-permissible timing, the next valid timeslot is computed and the difference in time is propagated to the start of the flight. This, in effect, forces the flight to start sufficiently late so as to reach the constrained airport at the desired time.

Thus whenever a flight is generated, it is ensured that there is no violation of slot and curfew times, if any. In other words, this constraint is treated as a hard constraint.

Chapter 7. Analysis of the Model and Questions

A number of questions arise in formulating and evaluating a perturbation-based approach for solving a complex problem like ATP. Many of these relate to the formalisation of the problem as a perturbation model, implications of the various issues and parameters on the quality of solution and impacts of the various alternatives in implementation, on the execution speed, and so on. In this chapter, we formulate some of these questions, consider the issues involved in answering them and hypothesise answers for them. In Chapter 8, we perform detailed empirical measurements and analysis to evaluate these hypotheses.

The questions we pose are the following:

1. Perturbation operators play an important role in a perturbation model. However, random changes [10](perturbations) in a feasible schedule may result in infeasible schedules. What are the issues to be considered and what options are available for identifying useful perturbation operators? How does the selection of such operators affect the system performance in terms of run-time and quality of solution obtained?
2. What is the desirable population size to be used? What is the significance of large and small population sizes? Does it depend on the data set used? What parameters affect the choice of a population size?
3. As we move from one generation to another, what should be the strategy to carry forward the population? In a broader context, what

[10] Such as replace one aircraft by another, change the route of a flight, etc.

are the issues and options for population management? What are the effects of various options here?

4. What kind of convergence behaviour is obtained? How does convergence depend on the various parameters used for the model? What is the effect of local optima on convergence?

These questions are discussed one by one in detail in the rest of this chapter.

7.1 Perturbation Operators

As mentioned in Section 6.3, there are two types of perturbation operators possible – we will call them crossover operators and mutation operators as per the GA terminology. From an analytical perspective, crossover operator provides for better exploitation of currently available solutions and mutation provides for exploration of new avenues in the search space.

Exploitation is concerned with making use of the solutions we have at any time and finding how best they can be exploited. Thus we combine parts of known good solutions expecting to get better solutions. No new avenues are explored here. Whatever structure the resulting solution possesses, will be borrowed from one of the parents. This generally leads to faster convergence, since new options are not open for trying. Also if most of the elements in the population converge to one solution, crossover becomes unable to produce a solution different from its parents. In almost all cases of crossover operators reported in the literature, the offspring from crossover of two very similar parents is not much better than the parents.

Exploration provides for moving around in the search space looking for unseen peaks. Mutation operators, which make random changes to a given schedule, address this objective. Exploration is essential to reduce the impact of local optima and premature convergence.

A balanced mix of exploration and exploitation is essential for good search performance by perturbation-based models. Full exploration with zero exploitation results in random movement in the search space and therefore, poor search performance. Note that the Darwinian selection process of favouring better solutions to survive into the next generation also provides for exploitation.

7.1.1 Crossover

The basic premise of crossover is to borrow features from the parent solutions and to use them to produce offspring solutions. In case of linear arrays or bit-strings, the common practice is to split both the parent solutions into two (or more) parts each, and to generate a child solution by interleaving the parts from the two parents alternatively or stochastically.

Since the time table structure is complex and inter-dependent, it is difficult to visualize many usable crossover operators. If the crossover leads to solutions violating any hard-constraint, such solutions have to be discarded. This is very expensive without a clear control over the percentage of such bad solutions likely to be generated. Most parts of an airline time table are interdependent for reasons such as sharing overlapping target-load (for example, a passenger group with the choice of using any one of two flights), using a common aircraft, etc. The wide variety of compatibility constraints makes this problem harder. For example, consider a simple crossover where for a given aircraft, flights are borrowed partially from one schedule and the rest from the other. This is very likely to lead to discontinuous flights (flight from A to B followed directly by a flight from C to D), and thus to infeasible solutions.

Apart from the sharing of the load table, schedules of two aircrafts can be considered fairly independent. Based on this observation, we have formulated one type of crossover operator, which is based on swapping of schedules assigned to different aircrafts. Consider two schedules S1

and S2, and an aircraft v. We replace the vehicleschedule that S1 currently contains for this aircraft v, with v's vehicleschedule in S2. Since vehicle v was able to execute the full movement sequence as part of S2, it can do so, in S1 as well. Therefore, this does not introduce any

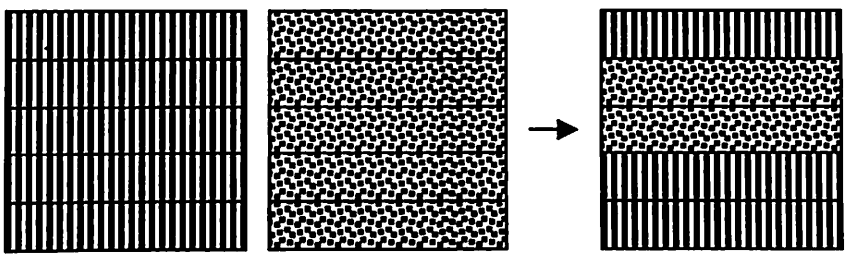


Figure 7.1: Crossover of Schedules S1 and S2

incompatibility or hard-constraint violation. This process is repeated for every vehicle. In effect, the resulting solution is a random interleaving of the vehicle schedules of S1 and S2 (for example, see Figure 7.1). We generate only one new child from a randomly selected pair S1 and S2.

7.1.2 Mutation

Mutation, though comparatively easier to visualize than crossover, produces its own problems. The typical operations one can consider for local modification of a time table are: change the vehicle (aircraft) of a flight, delete a random flight, insert a new flight at a random point in a vehicle schedule, etc. Changing a vehicle for an arbitrary flight is not possible, since compatibility has to be checked and also the flight must fit with the other commitments of the target aircraft. These are quite complex to enforce. Similarly while adding/deleting a flight also, we have to ensure that continuity of movement is maintained. Since the route patterns impose fairly significant movement constraints, this becomes difficult to enforce and will invalidate most options generated for mutation.

We have implemented mutation by choosing a random flight of a random aircraft from the schedule and deleting all further flights of that aircraft starting from that flight. Through a process similar to the

random initialization mentioned earlier, the vacant time of that aircraft is filled by adding fresh flights, as much as possible. Around this basic structure, a few variations have also been tried including:

1. Instead of selecting a flight randomly for deletion, bias the selection towards low-profitability flights. A roulette wheel of the profits of the various flights in a vehicleschedule is constructed and then a flight is selected based on it as the starting point for deletion.

2. Instead of deleting all flights from the point of deletion, delete all flights till the aircraft comes to the same base as before the selected flight. This will ensure continuity of movement and will reduce the "damage" done by mutation.

3. Stochastically determine the extent of deletion – a balance between the shortest segment that can be deleted and deleting all the flights till the end. The config file parameter (see Section 6.2) "del_stop" controls this behaviour.

4. Insert new flights at the point of deletion, rather than doing so only at the end. The config file parameter (see Section 6.2) "ins_prob" controls how often insertion will be attempted during mutation. So far, whenever we deleted a sequence of one or more flights, the gap was filled by advancing the timing of the subsequent flights of that aircraft, and new flights were added only at the end of the set of flights scheduled for that aircraft. In this variation, we attempt to insert one or more new flights in the gap created by deletion. Please note that the duration of flights inserted in place need not be equal in duration to the flights deleted. The timing of all flights are freshly computed every time a change is made in the schedule (internal optimizations minimise the resulting overhead by modifying only those flights which are likely to be affected).

Studies (for example, [Mitchell et al, 1991]) show that, for general genetic algorithm approaches, crossover is critical for efficient convergence to a solution as it paves for the combining of chromosomes from better individuals in the population. Mutation, in many GA approaches, plays a subsidiary role focusing on providing genetic diversity in the population and preventing premature convergence. However, this notion is being questioned [Spears, 1993], particularly as most of the GA approaches move away from binary string representation to problem specific data structures for representing chromosomes. There are contradictory results from researchers supporting and opposing the dominant role of crossover.

However, in a domain such as scheduling the argument favouring crossover is not likely to hold good. The main reason is that, for crossover to achieve the intended effect, the different parts of the chromosome must be largely independent. This is not the case, when the chromosome represents a complex structure such as a schedule. It is also necessary that most of the solutions generated by crossover do not violate hard constraints, as this will make the resulting solutions poor[11] in quality. The strong inter-dependence of the schedule structure makes this a difficult requirement to satisfy. Therefore, one expects a much less significant impact for crossover in the performance.

Thus, mutation plays a much more significant role here, than is normally implemented in traditional GA models. Also, in a perturbation model, mutation (i.e., perturbation) is the major player and hence is expected to contribute better.

What are the requirements that a good mutation operator must satisfy?

---

[11] Violation of hard constraints is fatal for a candidate schedule, while violation of soft constraints attracts only penalties.

1. One may be the ability to reach far and wide in the search space. An ideal mutation function should be capable of transforming any given solution to any other solution through a series of changes. Without this property, certain parts of the search space may become inaccessible, given an initial set of solutions.
2. Second characteristic would be the feasibility of the resulting schedule. If the mutated (perturbed) schedule is likely to be infeasible – violating any hard constraint – then either the schedule has to be discarded or it has to be repaired. Both these options negatively impact the efficiency of mutation.
3. The third aspect would be the ease of implementing the mutation operation, and its runtime complexity.

Considering these three aspects, we would expect the four variations of mutation proposed earlier to have quite similar behaviour, in general. Restricted deletion tends to make less drastic changes to the schedule and hence should perform better. All of the operators are implemented so that hard constraints are not violated in the resulting variations.

The reachability in the search space has two dimensions.

First dimension is of connectivity. Is it possible to transform any given schedule to any other schedule via a series of mutation operators? Our formulation of mutation operator ensures that this is satisfied. Consider two schedules S1 and S2. Based on our structure for representing schedule, both will have the same set of aircrafts. Therefore, difference between S1 and S2 can be transformed to differences between the vehicleschedules of the corresponding aircrafts. The vehicleschedules being a sequence of flights, we can transform any vehicleschedule V1 to another vehicleschedule V2, by examining the various flights in V1 one by one. To start with, set current flights in V1 and V2 to be their respective first flights.

- If the current flight in V1 is identical to the current flight in V2, then move to next flight in both V1 and V2.
- Else identify a subsequence of flights to be deleted from V1 starting at the current flight (in our case, this will be till we reach a base location).
- Now if V2 reaches the same base location somewhere in its schedule, replace the deleted segment by the set of flights in V2 from current to the base location. The number of matching flights between V1 and V2 increases.
- If V2 does not go by that same base location at all, the marked segment is deleted. This brings another flight at the same point for comparison against V2. Note that whenever deletion of flights takes place from the schedule of an aircraft, new flights are added at the end of its schedule. Eventually we expect a similar route as followed in V2 to appear in V1, increasing the amount of matching flights.

For example, consider V1 as (2,1,4,8,6) and V2 as (1,4,10,4,11), where the numbers indicate the route-id. Id 4 may represent Mumbai-Amsterdam-Paris, for example. The numbers chosen here are arbitrary and has no mapping to any dataset used; it is only to illustrate the transformation process. The aim is to transform V1 to V2. Here is one possible sequence as per the algorithm given above:

V1 (bold parts indicate the extent of match)	**Comments**
(2,1,4,8,6)	Mismatch at 2. Delete 2. Assume 10 got added.
(**1,4**,8,6,10)	8 mismatches. Delete 8 and 6. Assume another 4 and 8 got added at the end.
(**1,4,10,4**,8)	8 mismatches. Delete 8. Assume 11 got added.
(**1,4,10,4,11**)	Full match.

Thus, a series of mutations – perhaps a long one – can transform any given schedule to another. Note that the arrival-departure timings are computed by the system, and hence need not be matched in the transformation stage.

The second dimension is the path of such transformation. This is an issue because, if during the transformation, we need to go through lower profit states, the path is very unlikely to be taken – since given the higher profit of the parent state, the children will be removed in preference to the parent. Given two schedules, we expect the different operator variations given earlier to differ in terms of the average length of the path required to transform one to the other. For example, in our connectivity proof, we used the possibility of replacing a flight sequence by another; this is possible if insert_flight option (option 4) is used. If this provision is not used, the number of moves required for a transformation may increase in some cases. We expect this aspect to result in differences in performance among the various options listed above.

We carry out empirical study to test the performance of these variations on a number of datasets; results are reported in Chapter 8.

7.2 Impact of Population Size

A population of size one reduces the perturbation model to a standard hill-climbing algorithm with the consequent high risk of losing the best solution as well as of getting stuck in a local maximum. On the other hand, if an infinite population size is used, the optimal solution will be present in the initial population itself. For finite population sizes, the performance must obviously be something in between.

Do the convergence and/or the quality of final solution reached, show a monotonically increasing dependence on the population size? Or will it taper off after a certain population size is reached?

For very small population sizes, the chances of all chromosomes converging to a common solution – perhaps prematurely – is high and hence convergence will be fast, but the quality is likely to suffer. Assuming a randomly selected initial population, a larger population size offers a wider coverage of the search space, and therefore, better chance of getting to the optimal solution. However, we would like to keep the population size small to reduce the run-time. Therefore, we would like to avoid very large population sizes.

We measure the average solution quality reached after a fixed number of iterations for a variety of population sizes and report in Chapter 8.

One related issue is the comparison of higher population size vs larger number of iterations. Both these options are equivalent in the sense of providing for wider exploration of the search space. Both higher population size and higher number of generations increase the number of schedules tried during a run. So, it would be interesting to know how do these two parameters interact? Do increasing population size and increasing the number of generations produce similar effect on the

solution quality? We return to this interesting issue in Section 7.4 of this chapter.

7.3 Population Management Strategy

Given a set of schedules – representing a random collection of points in the search space – in each generation, we generate fresh schedules using the available operators. In order to keep the number of solutions bounded, as new schedules are generated, we need to discard some of the schedules from the resulting pool. There are a number of factors that affects the choice of a suitable strategy for this population management. These factors include:

1. Ensuring that the best solution(s) found are not lost; since the search is stochastic, we are not guaranteed to rediscover them within our time limit.

2. Providing higher chance for better solutions to survive – essence of a Darwinian style of evolution.

3. Providing adequate opportunity for new solutions to be derived from the better solutions that we have (exploitation).

4. Providing adequate freedom for new areas in the search space to be explored (exploration).

A suitable strategy needs to balance/address all these concerns.

Early models of GA experimented with replacing the entire existing pool by the generated solutions. This was found to be a poor strategy since, if the children happened to be poorer than the parents, good solutions are lost from further explorations. Therefore, almost all perturbation and GA models use an elitist strategy, where the best few schedules are carried forward to next generation. In addition, a fraction of the current population is retained, and the rest are discarded. There are two questions here: (a) what should be this fraction of the current population to be retained? and (b) how should we select the schedules

to be retained. We call the fraction of schedules to be retained as the **persistence factor** and will be denoted by p. Thus p denotes the fraction of schedules retained from one generation to the following generation. We will denote p as a percentage. As p increases from 0% (nothing carried forward) to 100% (nothing new generated) what can one expect?

At p=0, we have a purely random search through the search space, generating completely new random schedules at each generation. Therefore, we will ignore the case of p=0. Similarly trivial is the case of p = 100, where the search stagnates after the first generation. As p increases from 0 towards 100, more and more solutions are retained and two trends can be noticed:

1. There will be more solutions from which parents can be selected for crossover and mutation, during perturbation. Hence, the scope for crossover and mutation increases, resulting in better search.
2. The number of new schedules to be created reduces, thereby reducing the extent of search.

These are two opposing effects on the average solution quality from generation to generation. At the two extremes, one or the other of these two trends dominates, affecting the progression of quality negatively. Thus one expects a plot of solution quality against p value to show a peak somewhere in the middle and flatten out at low and high values of p.

The second question is how should this p-fraction of the solutions be selected? The simplest strategy perhaps is to retain the best p% and discard the rest. We will denote this as strategy s1. One problem with this strategy is very early convergence. As one or two solutions approach a local maximum, they will begin to dominate the population. Due to the larger profit value, they get higher chances of being selected

for mutation and crossover. The resulting offsprings are likely to be located close to the parents in the search space. Solutions on that local maxima fill up the pool fairly quickly.

Therefore, sustaining some part of the currently not-so-good solutions may be useful, as it may lead to a different (and possibly better) peak. We test this out as strategy s3. In s3, we keep the best p/2 %, and then choose another p/2 % elements randomly from the rest of the pool irrespective of their profit value. A variation of this strategy, under the name s2, is also implemented where we keep the best p/2, select p/4 from the next p/2 (randomly), and the remaining p/4 will be the best of what remains. These bias the selection towards the better solution, ensures that the current best solution is not lost and still provides a chance for other solutions to survive for a while.

Statistics suggests use of mean and standard deviation as another perspective relevant to this kind of problem. A simple idea here would be to discard those solutions which lie beyond the point, mean – k*σ, where σ is the standard deviation and k is some constant. Normally preferred values of k are 1 or 2. We implemented this strategy as s4 for different values of k.

One can also consider introduction of fresh solutions into an existing pool, to reduce chances of premature convergence. This has been implemented as strategy s5, where after retaining a fraction p of the current population, a fraction q is created as fresh random solutions, and added to the pool. For retaining the fraction p, strategy s3 is used. The balance (1-p-q) fraction is generated using the mutation and crossover operators as before.

We can reason about the performance of these strategies as follows. The essence of perturbation model is a proper blend of exploration and exploitation. The strategy most likely to yield dividends is one that can provide that balance. s1 is simple and ensures exploitation – but provides little support for exploration. When population size is large,

even retaining only the best few can provide some spread of solutions; however the spread is likely to be very small. As the solutions converge, even this possibility reduces to zero. s3 provides adequate exploitation retaining p/2 best solutions. In addition, by providing a random subset of the remaining solutions to survive into the next generation, adequate scope for exploration is provided. Hence we expect s3 to perform better than s1. s2 should, by the same rationale, produce a result similar to s3.

S4 is more difficult to rationalize. As we saw earlier in Chapter 5, the solutions for ATP follow a Normal-like distribution in the space. Average and standard deviation are obviously a function of the values present in the population at a given time and the spread of these values. We experiment with a range of k from 0.25 to 2.0 to explore the variations. Strategy s5 also seems difficult to analyse without some performance results.

Section 8.6 presents experimental results on the effectiveness of all these strategies.

7.3.1 Handling Duplicate Solutions in the Population

Another aspect of population management is handling of duplicates generated in the stochastic exploration process. Since the search is stochastic, it is not possible to guarantee that duplicates will not be generated. Duplicates can occur in two forms: identical solutions and solutions with same profit. Since, the profit uniquely determines the quality of the solution, identical profit can be considered as a duplicate solution as far as selection for survival is concerned. Retaining or eliminating duplicates need to consider the following issues:

a) Checking if a solution is identical to one of the existing solutions is expensive. This can be optimized to some extent, by first checking if the profit matches and then only going for a detailed comparison.

b) Solutions with the same profit may or may not be identical. If they are not identical, discarding such solutions based on profit-match, may throw away better candidate solutions. On the other hand, since our measure of quality is the profit, we may take the view that if there are two different solutions with the same profit, we may accept any one of them.

c) Keeping true duplicates reduces the effective population size and may also contribute to premature convergence by encouraging further duplicate solutions to be generated from them.

d) If we discard duplicates, we will either need to generate additional variants to meet the desired population size, or accept a reduced population size.

We also study the effect of retaining or discarding duplicates on the solution quality; results are reported in Section 8.6.1 .

7.4 Convergence of Solutions

As the generations progress, how does the solution behave? Does the entire population converge to a single (local) optimum? Or does the population retain adequate divergence? How do the various parameters discussed earlier influence convergence?

When crossover is the primary perturbation operator, convergence is often found in most cases. When mutation plays the dominant role, one expects this not to be the case. Probabilistically, as per the mutation operator defined earlier, a very large number of new schedules are possible from a given schedule via mutation.

Please recall that mutation selects a vehicle schedule, and then a flight and deletes a set of flights around this selected flight. Considering the deletion itself, there are |V| * F possible deletions from a given schedule (F being the average number of flights by an aircraft). In all

cases following a deletion, one or more flights are added. Considering a single flight being added, there are |R| options possible, where R is the set of routes. Thus, the number of possible schedules that can result from a single mutation on a given schedule are of the order of |V|*F*|R|. Thus, we do not expect complete convergence of a population to a single value to happen, frequently.

There are many options for mutation to generate a spectrum of perturbed solutions. However, recall the profit distribution we have observed in Chapter 5. As the profit of the current solution increases, the probability of mutation generating better solutions decreases exponentially, irrespective of the number of options available. This is further compounded, if the mutation operators are restricted in the region it can reach from a given solution. Thus, as the number of generations increases, new mutated solutions that survive to next generation may be largely replicas of existing ones and therefore, we may observe convergence of the population even in cases where mutation is the dominant operator.

We hypothesized the relevance of population size on the solution quality, earlier in this Chapter. We also know that a number of generations are required to reach good solutions. What is the relation between population size and the number of generations required to reach a given quality of solution?

Note that as more generations are created, mutation gets more opportunity to explore more options. Similarly as population size increases, more area of the search space can be sampled. Therefore, at some level, we expect the quality of solution to increase with increase in population size, as well as with increase in the number of generations.

But does increasing generation count produce the same effect as higher population size? For answering this, one need to look again at the solution-quality distribution graph discussed in Section 5.2. Observe

that most solutions in the search space are of medium quality. Therefore, most of the solutions initially created randomly are likely to be of this class. The probability of getting a higher quality solution through mutation of a solution here is proportional to the fraction of the area under the curve to its right.

However, as the generations progress, due to natural selection process, the average solution quality in the population would increase, moving the point further and further to the right. The probability of producing better solutions decreases as a result.

Whereas, choosing a larger population allows a wider coverage of the search space to start with and hence directly yields a higher chance of finding better solutions.

Thus one expects that running a model for a larger number of generations is not likely to be a cost-effective alternative to using a larger population size.

7.5 Cooperative Time Tabling Across Airlines

The presence of competitors aiming for a common pool of passengers introduces a substantial element of uncertainty in one's planning of time tables. Incomplete knowledge of the competing airlines' strategy further complicates demand estimates. Whatever optimisation is done by one airline would not be fully effective unless it is coordinated with the schedules of other airlines. One interesting direction of thought here is to consider the various competing airlines joining together to prepare a single common schedule, pooling all their resources and addressing the net demand estimated -- a combination of competition and cooperation to be called ***cooptition***. Considering the various subjective and objective constraints (including available aircrafts, carrying capacity, operating locations, etc), the flights in the resulting schedule can be divided among the various participating airlines. Such an approach makes the pay off in automated scheduling system much

more worthwhile and makes the quest for the optimal schedule more appealing. This sketch may make the whole problem seem simple. Let us dwell on this in a bit more detail.

Imagine two major airlines flying in a common region and working together to prepare a jointly optimized schedule as mentioned above. How do they divide the flights in the schedule among them? As one approach, they could put together a set of "bids" for buying each flight from the pool. The bidder quoting the highest amount would win the bid and operate the flight. The payments to be made can be netted out and the balance could be paid by one airline to the other. The bid prepared by an airline will be dependant on its investment in that sector, resource availability, utilization of resources such as crew and how they integrate with the rest of the operations of the airline, etc.

The airlines may also agree in advance to have certain sectors to be reserved for each one, to be serviced only by specific types of aircrafts, etc. This kind of arrangement may reflect the respective airlines' linkages with other connected sectors/airlines, perceived market value in that sector, etc.

One can expand this model to more than two airlines. It is also possible for one or two major players to setup the basic cooperation framework and invite others to join in the cooptition process.

Although a flight-wise or sector-wise distribution is more amenable to a process like bidding, this introduces problems of logistics in the schedule. The flights gathered by an airline, may be spread across any arbitrary subset of aircrafts in the pooled schedule. To ensure usability of the schedule, it would be desirable to distribute the schedule among the airlines based on aircraft. This also is not, by itself, a satisfactory solution since optimisation process normally does not attempt to distribute profit evenly among the aircrafts. This may result in net profit from all the flights operated by an aircraft varying significantly from

aircraft to aircraft. For example, compare an aircraft operating in the US sector with one in the Gulf sector.

Major practical hurdles make it difficult for airlines to accept this cooptition model readily. The issues go beyond market factors and organizational issues to the realms of management sciences and economics. It is a major scholastic challenge to show that compared to existing schedules, there exist coordinated schedules, which benefit everyone concerned including the airlines passengers. The fundamental principle here is that certain types of competition cause losses to all concerned and we are beginning to see technological support to activities turning such competition to a mutually advantageous cooptition.

At one level, it is easy to argue how a coordinated schedule would produce better results than two individual schedules. There are two obvious scenarios. Consider a medium demand sector with, say, about 300 passengers estimated. Divided across two airlines, one estimates a load of 150-200 at most. For typical aircraft capacities, one would schedule a low capacity aircraft or allow under-loaded flights or decide to abandon this sector. In a coordinated schedule, there is scope for a flight with normal size aircraft. This could be operated by one of the airlines as agreed before.

Another case is where there is a large enough load for both. What time do they choose to fly this load? Rather than make two flights at nearby timing, targeting peak load, one could stagger the flight sufficiently to have no load-drop-off at all, leaving a higher load factor for the airlines and more freedom and convenience to the passengers.

7.6 Transferability of the Principles

One question that an analysis, as portrayed so far, brings up in one's mind is "how generalisable are these analyses?". As mentioned in Chapter 2, there is increasing understanding among practitioners that a

universally successful optimisation method is unlikely. Fairly general frameworks such as iterative refinement, genetic algorithm, perturbation models, etc. have a wide range of applicability primarily due to their relatively low dependence on domain specific details such as type of objective function and nature of constraints. However, success of these techniques, wherever reported, is largely due to clever and careful choice of the many parameters and operators on a domain-to-domain basis. This approach has resulted in these techniques being portrayed as empirical and ad hoc.

In between the search for a well understood, completely general-purpose approach and ad hoc results in solving a specific instance -- both of which has little useful lessons to contribute -- we have adopted a middle path, namely, to develop a framework for a problem class of airline time tabling - not generalising to arbitrary scheduling, or further still, optimisation problems, and not specialising to a particular type or instance of airline. This is more likely to lead to useful results applicable to an important class of problems. This approach also enables us to study effect of various aspects of this problem class on the framework.

Thus our observations and analysis of the perturbation framework in the context of airline time tabling will be applicable to any airline time tabling problem. We have made no serious assumptions regarding the nature of the airlines operations in our problem definitions. Further, we expect much of the results to be significant in the next higher class of problems, that is, vehicle scheduling problems, since we have modeled airline time tabling as a vehicle scheduling problem.

Chapter 8. Results and Analysis

This chapter reports the various studies carried out and analyses the results obtained. Unless otherwise specified, the master component doing coordination and user interface was run on an IBM Pentium II processor with 128 MB memory running Windows NT. The slave, running the perturbation model, was setup on Pentium III, 800 Mhz system with 128 MB memory running Linux. The Java version used was 1.3.

8.1 Datasets Used

Two types of data sets were used for the analysis reported here. One type is based on the simplified model (STP) described in Chapter 5. These have been named as SIM1 to SIM4:

- SIM1: Low passenger demand. Total number of passengers in the dataset= 4600
- SIM2: Low passenger demand and non-uniform demand across city pairs. Eg: Alpha-Gamma and Delta-Alpha sector demands are only in the morning. Certain sectors are given very low passenger demands. Total number of passengers= 5700
- SIM3: Moderate demand numbers. Total number of passengers in the dataset= 9200
- SIM4: A variation of SIM3. Total passengers in the dataset= 7750

The second type is based on an airline of realistic complexity (about the size of Air India).

- ACT1: Total passengers in the dataset= 29832.
- ACT2: Total passengers in the dataset= 29932, a variant of ACT1.

8.2 *Variation in Performance Across Iterations*

We performed a number of runs (iterations) of the program to analyse the degree of variation observed from iteration to iteration.

Table 8.1 gives the best profit[12] (in million Rs) recorded after 0, 100 and 1000 iterations for 10 different runs using the SIM4 dataset. Mutation with variation 1 (As explained in the previous chapter, this refers to making a biased selection of a flight for one aircraft and deleting all flights from that point onwards.), and crossover was in force.

Run No	Initial profit	Profit after 100 generations	Profit after 1000 generations
1	9.065	13.935	13.935
2	11.230	13.215	13.800
3	9.820	13.700	13.800
4	9.615	13.160	13.340
5	9.220	13.885	14.425
6	9.560	13.455	13.670
7	10.020	13.585	14.140
8	9.445	13.085	13.860
9	10.200	12.825	13.765
10	9.635	13.740	13.900

Table 8.1: Behaviour of profit (in million Rs) against iterations for SIM4 dataset

[12] Note that since fixed costs are not counted, this does not mean actual profit. It is just revenue minus running cost.

Figure 8.1 shows similar information graphically over 50 runs of the problem, with a different configuration. The population size used was 50. The plot shows the profit of the best initial schedule, of the best schedule at the end of 100 generations, and of the best schedule after 2500 generations. The mean final profit was 14.3 million Rs and the standard deviation was 0.33 million.

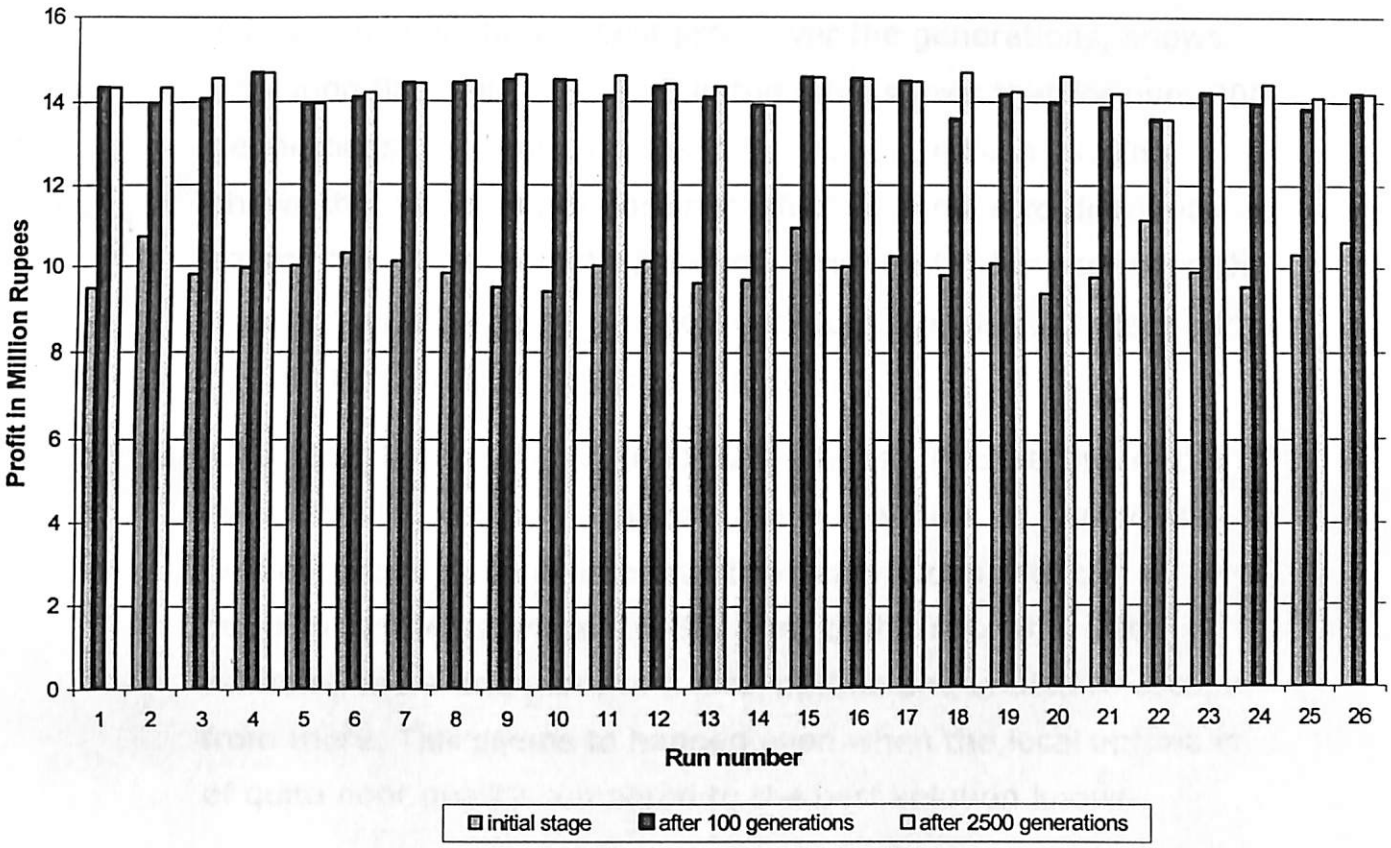


Figure 8.1 Profit improvement with generations for dataset SIM4 – for multiple runs.

8.2.1 Observations

From Table 8.1 and Figure 8.1 (Similar behaviours were observed many times with other data sets and different parameter settings during the experiments. The table and the graph are shown as representative examples.), the following observations can be made:

1. There is significant tendency for the system to get trapped in local optima. Very rarely have two different runs produced identical schedules – even in terms of profit. In the graph, compared to the overall best value obtained (14.880 million), in a significant number of runs, the best value obtained is as much as 10% lower than this. The fairly high figure of 0.33 million for standard deviation also projects a similar picture.

2. Tracing the variation of best profit over the generations, shows very long flat regions. Case 3 in the table shows that for over 900 generations, the profit increased by only 0.1 million Rs. This shows that the operators are not effective enough to distribute search across the search space regularly. Particularly so when the search gets trapped near a local optimum. This is consistent with our analytical observations in Chapter 7.

3. Also note that the convergence is quite fast. Irrespective of the initial starting value, the system reaches almost its final value (within 1 to 5 percent) in about 100 generations with a population size as low as 10-50. After that it seems to get attracted more and more to a local optima and unable to escape from there. This seems to happen even when the local optima is of quite poor quality compared to the best solution known.

We take a look at another dataset with only the mutation operator enabled, for the full-data set (ACT2). The program was run for 1000 generations, with a population size of 100. Figure 8.2 shows the profit of the best solution after 100, 500 and 1000 generations in each run.

Similar behaviour as before can be observed here also. Convergence here requires about 500 iterations, beyond that there is very little increase in the quality of the solution.

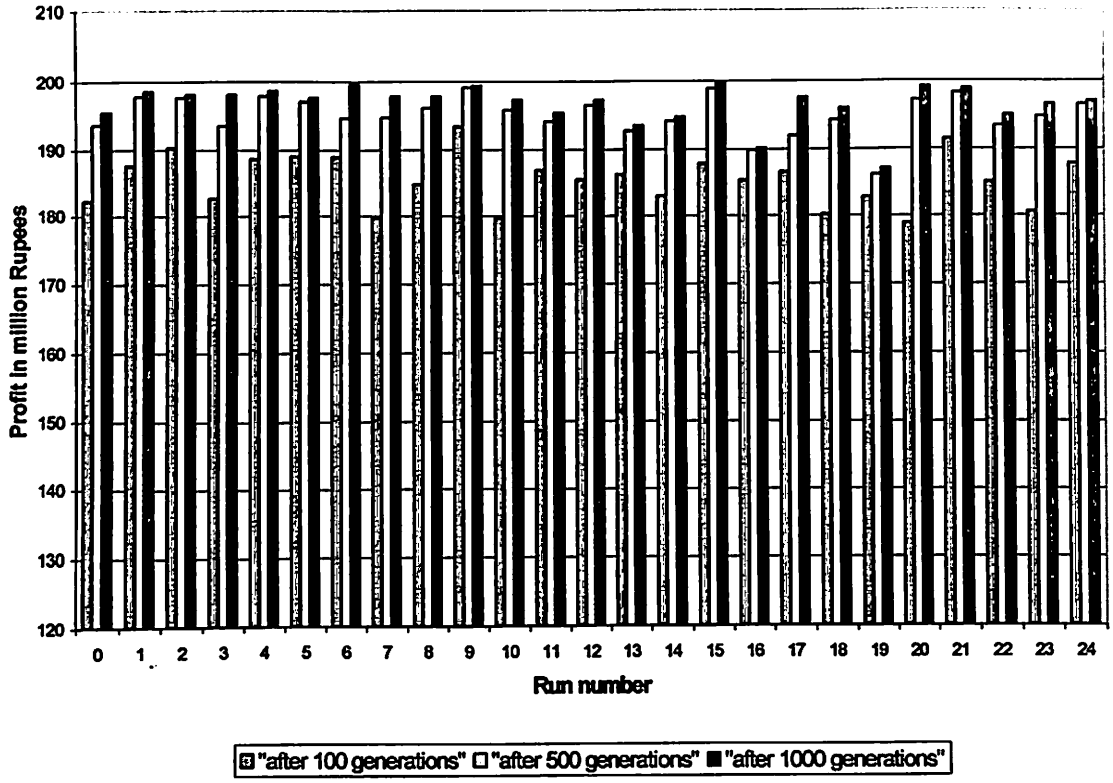


Figure 8.2 Profit improvement with generations for different runs of the dataset ACT2

The iteration labeled 19 shows a particularly bad case of local optima, where the resulting profit is about 187 million against the best solution known of 200 million. While such pathological cases are rare, they do occur. This, incidentally, is the reason for opting to use average final profit across a number of runs as the yardstick for analysis in our further studies.

8.3 Impact of Perturbation Operators

As mentioned in an earlier section, we have primarily concentrated on mutation operator as the primary perturbation operator. We did implement one type of crossover as outlined in Section 7.1.1 . In this section, we first report results of studies comparing the performance of the "SIM1" dataset on various types/combinations of operators. All runs used a population size of 50. The plot in Figure 8.3 shows the average profit at the start and the average profit reached at the end of 500 generations for the following operator combinations:

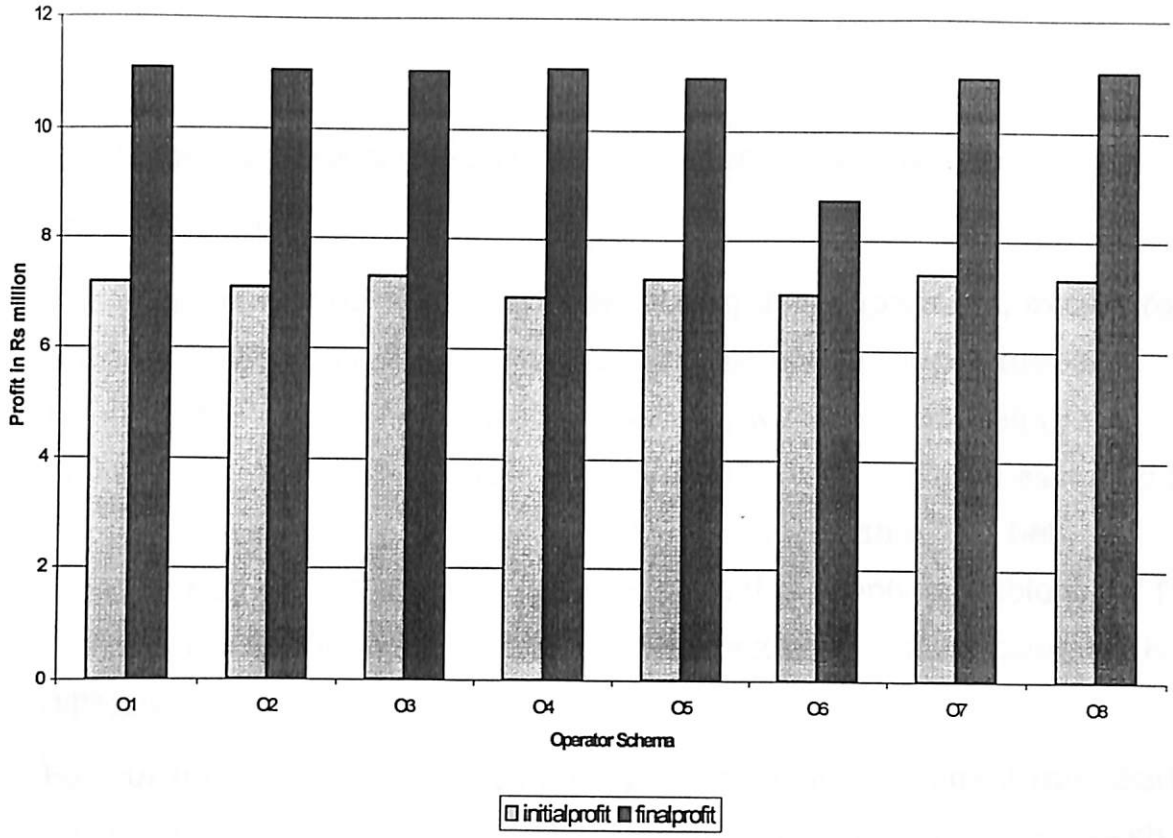


Figure 8.3 Average Profit after 500 generations under various operator variations (dataset: SIM1)

O1. Mutation with one roulette wheel for selecting the aircraft, and a second roulette wheel for the flight to be deleted. Roulette wheel enables a biased selection to be carried out. The bias in both the roulette wheels here are towards selecting the one with the lowest profit. In case of aircraft selection, the net profit of all flights operated by that aircraft is considered.

O2. Mutation with roulette wheel only for selecting aircraft, with flight selection being random.

O3. Mutation with roulette wheel only for flight selection, with random aircraft selection.

O4. Mutation with no roulette wheel; selection of aircraft and flight are both random.

O5. Both roulette wheels enabled, disable random insertion of flights at deleted points.

O6. Crossover only

O7. Mutation (best of O1-O4) and crossover at 0.5:0.5 ratio

O8. Mutation (best of O1-O4) and crossover at 0.2:0.8 ratio

Observations

The graph shows no major variation among these operators, except for a clear verdict against crossover as the main media for perturbation. We have tried only one crossover operator, which replaces all the flights assigned to one aircraft in a given schedule by flights assigned to the same aircraft in another schedule. It is conceivable that better crossover operators could be invented; but this is only possible work for the future. Our finding is that inventing effective crossover operator is difficult.

For our model, mutation plays a very essential and dominant role. Early perturbation models based on the popular genetic algorithm approach relied heavily on crossover as the primary vehicle for solution improvement. This was verified and reported in many studies leading to an overwhelming support for crossover and with mutation being used with a very low probability (often less than 1%) mainly to prevent premature convergence and to provide some genetic diversity.

However, analysis shows that this observation does not apply when the solution representation is not in the binary string format. Crossover is quite complex to define in most non-binary formats; many studies have used specially devised crossover operations to overcome the difficulties. Mixing parts of two or more solutions to produce a feasible solution is difficult when the solution parts are heavily inter-dependent, as is the case in our problem.

Thus, the crossovers in most such domains tend to be fairly weak, leading to its relatively poor performance. We see that phenomenon here. However, a properly utilized crossover does contribute to reaching the solution faster. See Figure 8.4.

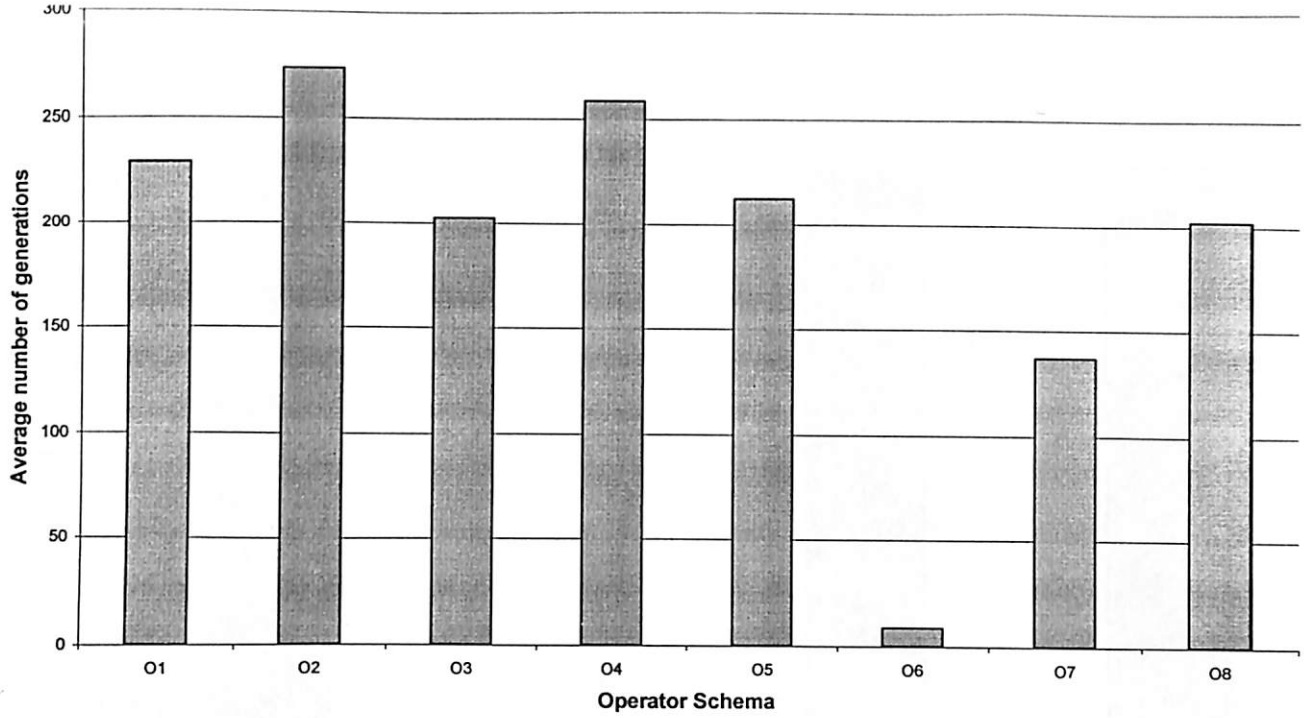


Figure 8.4 Number of generations used before reaching the final profit value for different operator variations. (dataset SIM1)

This graph shows the average of the number-of-generations after which no profit improvement was noticed during the same study. Note that as reported in an earlier section, the convergence is fast in general – taking only about 200-250 iterations in most cases. In the case of option O6 (only crossover), the convergence happens very early. This is no surprise, since our crossover is incapable of producing any new flight schedules for a given aircraft. Therefore, the best options from the initial random schedules generated are quickly identified and retained.

The important point to notice is the in-between values reached for options 7 and 8. The difference here is the partial presence of crossover. Note from the previous graph that the solution quality reached in these cases is quite comparable to that in other options 1 to 5. However, the solution is reached in significantly smaller number of generations.

We repeated the study with the full-data set, ACT1. The result is shown in Figure 8.5.

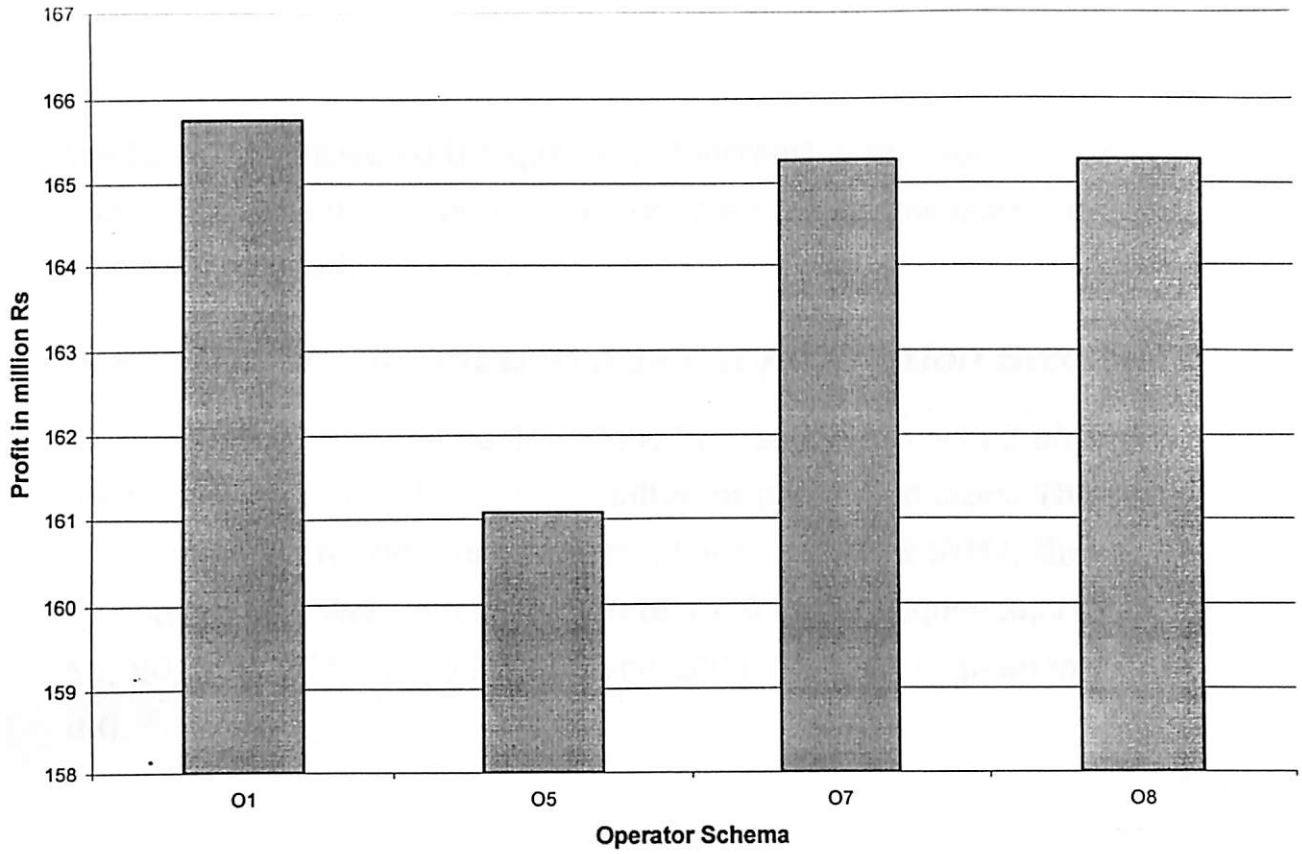


Figure 8.5 Final profit (in million Rupees per week) against operator variations (dataset: ACT1)

In this, we have focused on a subset of the operator schemes only. The graph shows poor performance for O5 where insert-flights have been disabled. Clearly the ability to insert flights enhances the power of the mutation operator significantly. This trend was not visible in the simple datasets; perhaps because the flight options there are very limited and therefore, the insert may not make as much an impact as in the case of the full dataset. Another factor contributing to this discrepancy is the comparatively similar duration of flights in the STP case, compared to the wide variation in flight duration (India-Gulf flight vs India-US flight) in the case of full data.

The relative insignificance of crossover is consistently reported in this graph as well. Crossover ranging from 0% to 50% (schema O7 and O8) produces no noticeable difference in the final profit reached. However, as in the case of the earlier dataset, we observed that the final profit value was reached in 10-15% less number of generations when crossover is present.

We have not answered the question of increasing population size vs increasing the number of generations. We take up this question after discussing the other issues.

8.4 Quality of Solution Against Population Size

This study analyses the quality of the best solution reached after a given number of iterations, using different population sizes. The study was conducted for different datasets. For the dataset *SIM4*, the population size was varied from 10 to 1000 in the sequence, 10, 25, 50, 80, 100, 200, 300, 500, 800 and 1000. The plot is given in Figure 8.6.

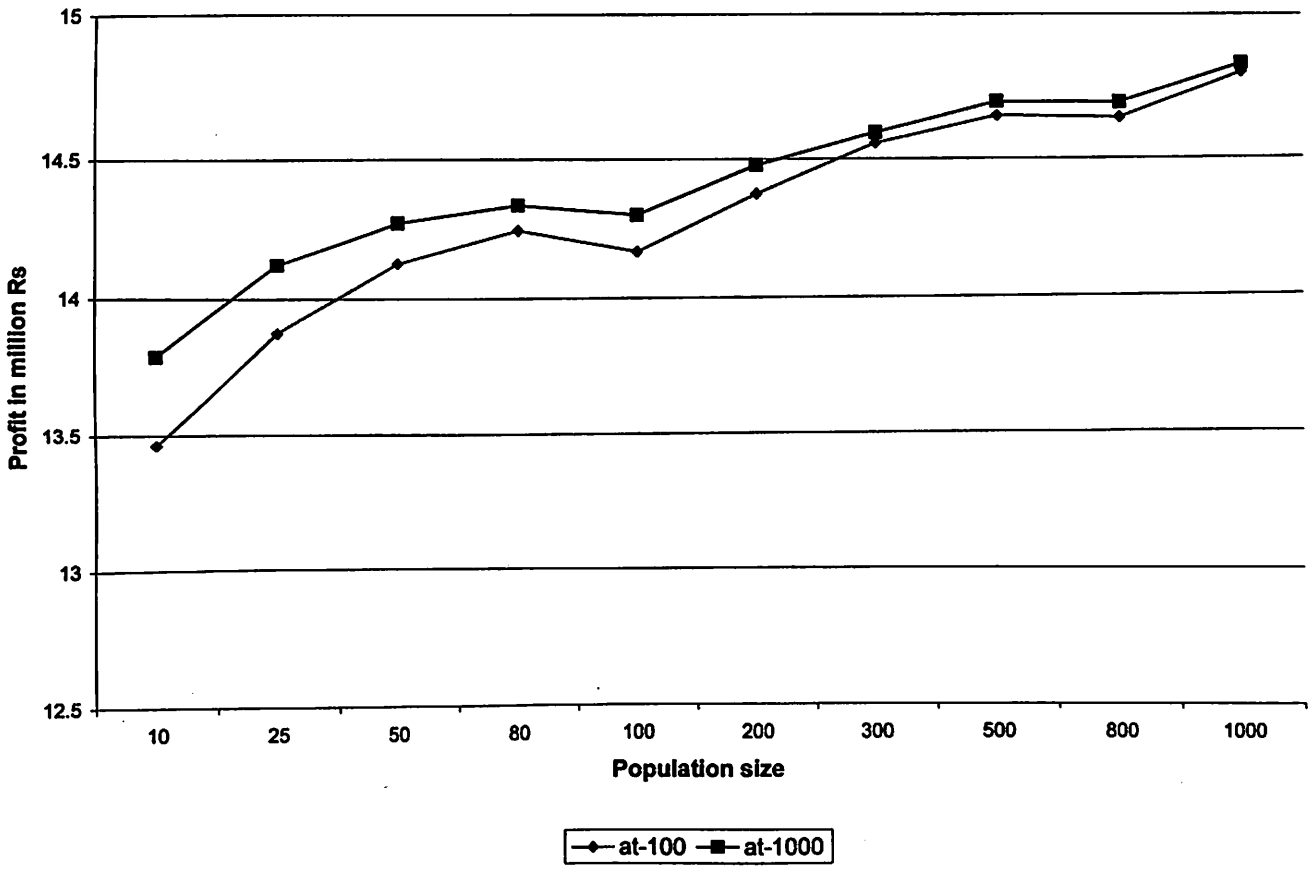


Figure 8.6 Average profit (in million Rupees per day) after 100 and 1000 generations against population size. (dataset SIM4).

The plot shows a visible increase in the average solution quality as population size increases. Compared to the average for a population

size of 10 (approx 13.8 million Rupees), the average for population size 1000 (approx 14.8 million Rupees) shows a 7% improvement in profit, which is quite substantial in these domains. (Also note that as mentioned earlier, since the fixed cost is not taken into account in our computations, the percentage gain here is a substantial underestimate.) The payoff relation against population size is not linear, though. As population size increases, the rate of improvement decreases. The uneven decrease seen in the graph is likely to be noise due to the stochastic nature of perturbation-based approaches.

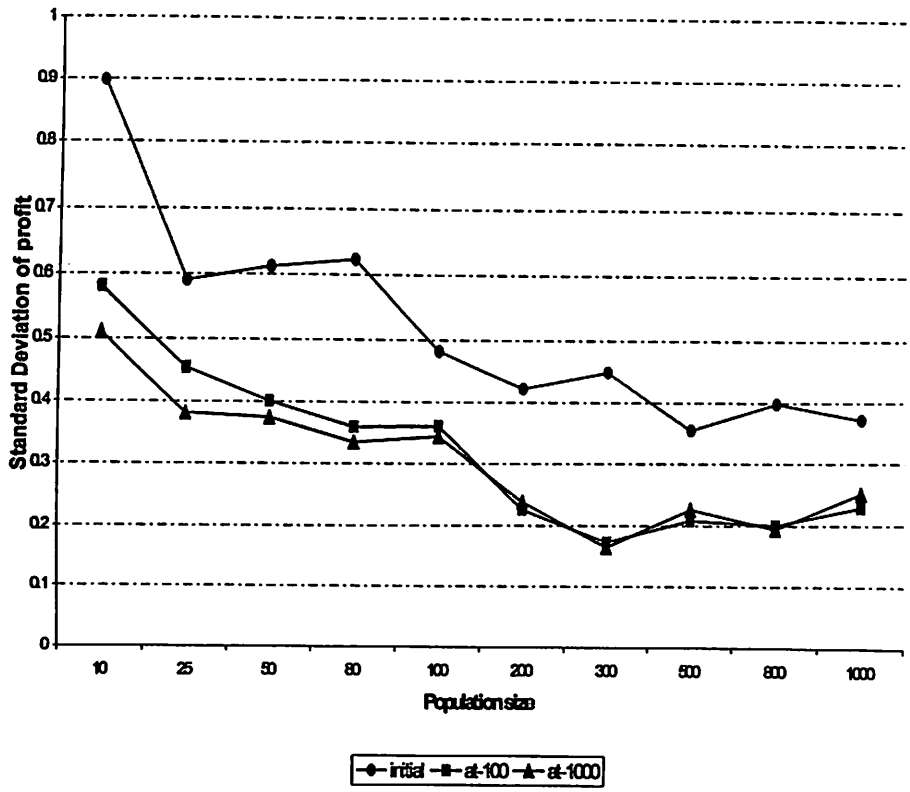


Figure 8.7 Standard deviation of profit of solutions, at various stages in a run, against population size. (dataset: SIM4)

It should also be noted that the standard deviation for the above series is substantial (shown in Figure 8.7) for small population sizes. Thus the variability of the best solution reached at the end of a given number of

iterations is high for small population sizes and stabilizes as the population size exceeds 100.

Thus the probability of getting high quality solutions in a given run increases as population size increases. A very large population size does not result in proportionately large improvement in solution quality. But, very small population sizes do tend to produce significantly poor quality solutions. Thus, a medium population size should be chosen to contain the high variance in solution-quality, as well as to ensure a fairly good probability of a good solution, and balance it against the high running time resulting from high population size.

One natural question here, is what should this medium population size be? In the SIM4 dataset, this can be seen to be about 100-200. We carried out similar experiments with a variation of the STP (SIM1) and the full problem (ACT2). The plots of average profit are given in Figure 8.8.

Note that this plot is also consistent with the observations made earlier. The quality improvement at the end of 1000 generations, as population size is changed from 10 to 200 is approximately 2.5%. A population size of about 200 seems to work well for this full problem.

The results from the dataset SIM1 (See Figure 8.9) again agree with the earlier result. There are a few important observations worth making here:

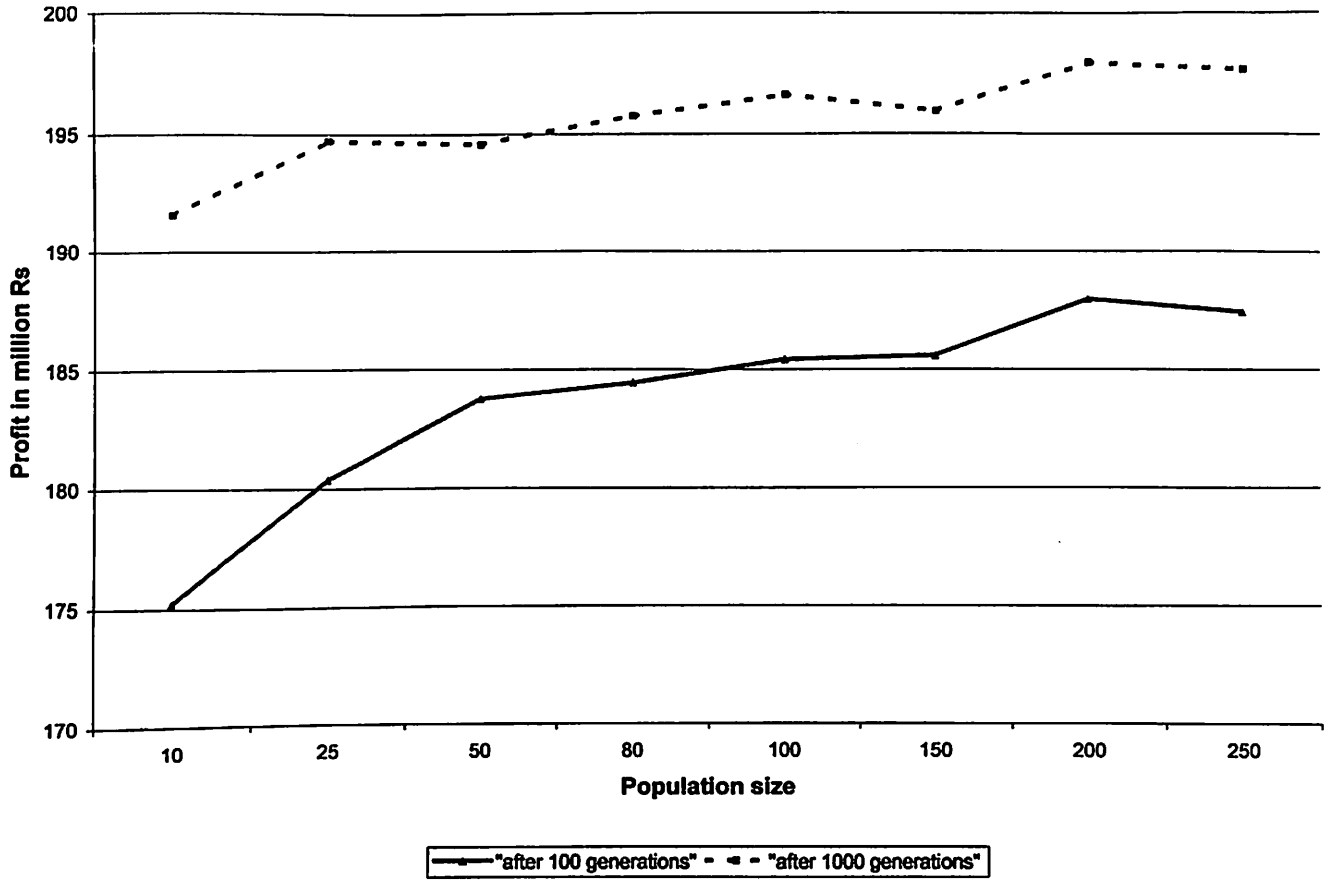


Figure 8.8 Average profit (after 100 and 1000 generations) against population size for dataset ACT2.

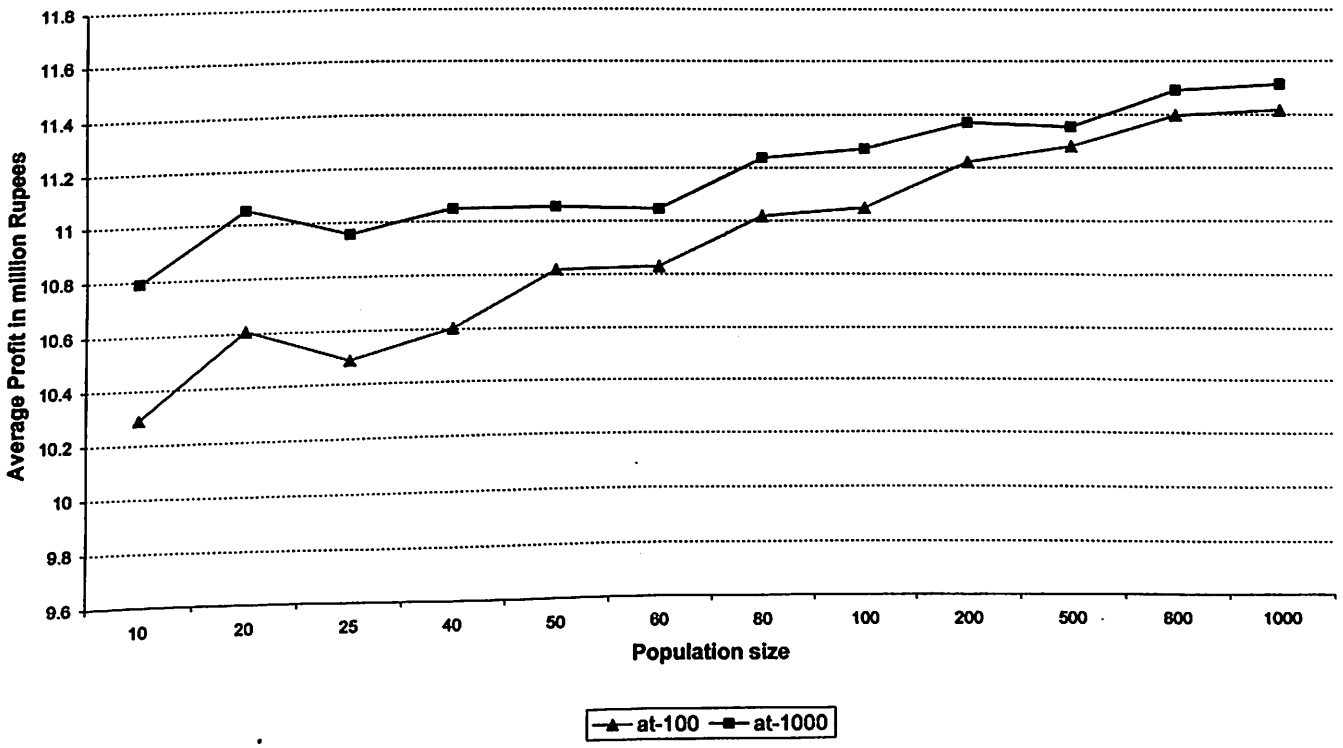


Figure 8.9 Average profit after 100 and 1000 generations, against population size for dataset SIM1.

1. The percentage improvement of quality achieved against population size is more significant at smaller number of generations. In the case of 'ACT2' dataset, the difference is about 7% at 100 generations which reduces to about 2.5% at 1000 generations. The difference was found to be almost this value at 500 generations as well. Thus, a larger number of generations partially compensates for smaller population size. However, the substantial difference even after convergence shows the significance of using sufficiently large population size.

2. The incremental improvement in quality decreases as the population size is gradually increased. Larger population size demands very high memory usage having to store and manipulate that many solutions at a time. For example, our machine could not sustain a population size over 200 for the 'ACT2' dataset. Therefore, it is important to choose a reasonable population size in actual runs.

3. While very low population sizes such as 10-50 produces visibly poor quality solutions, a dataset-independent 'recommended population size' is difficult to formulate. Based on our studies so far, a population of size around 200 seem to indicate an acceptable compromise between increase in quality of solutions and increase in run-time and memory. If larger population sizes are sustainable, it seems worthwhile to try, particularly for large or constrained datasets (ACT series and SIM1, for example).

8.5 Quality of Solution Against Persistence Factor

Recall that persistence factor, denoted by parameter p, measures the percentage of solutions retained from one generation to the next. For the data set "SIM4", the graph in Figure 8.10 shows the average profit reached after 100 generations, and average profit reached after 500 iterations, for different p-values, ranging from 0 to 100.

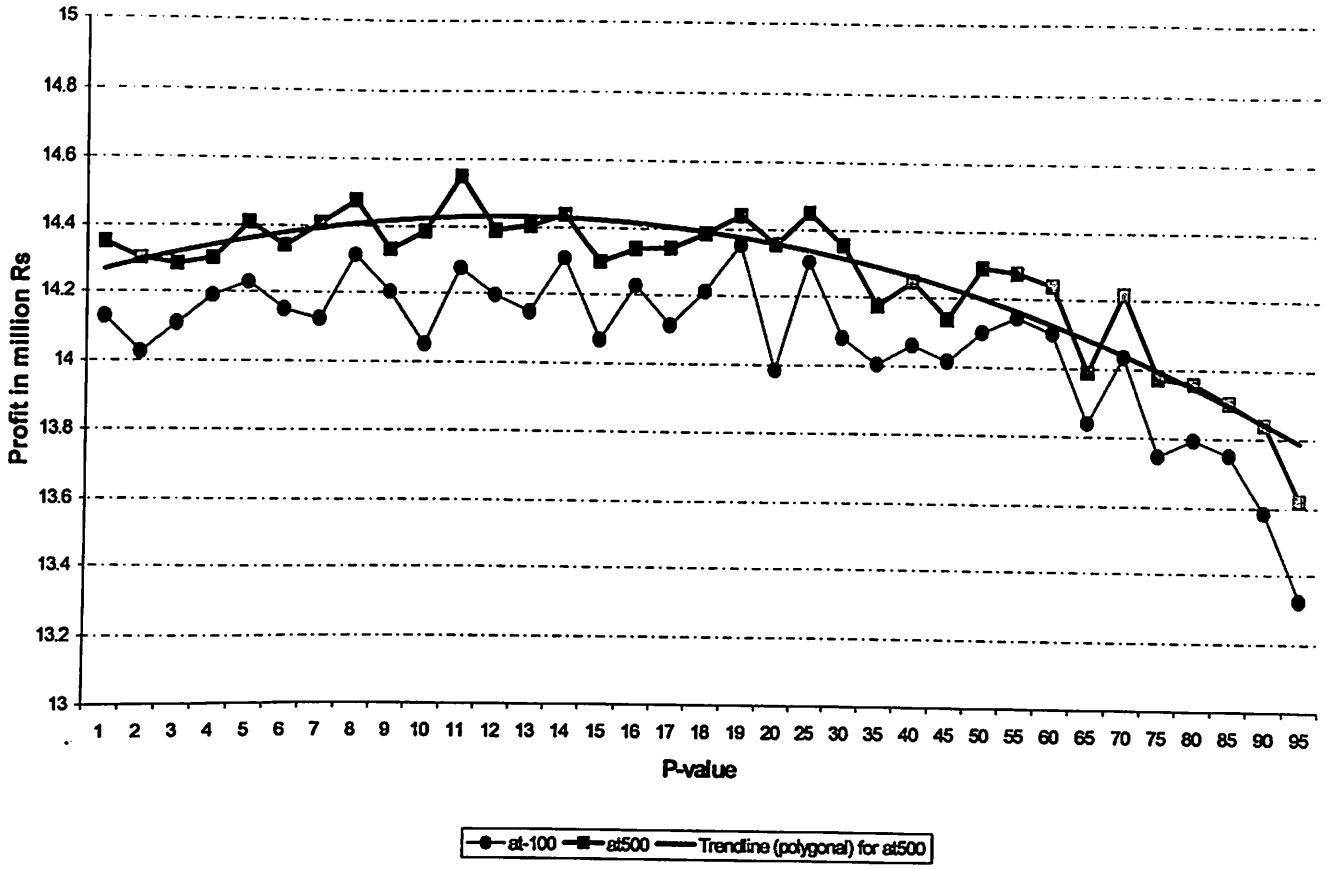


Figure 8.10 Average profit after 100 and 500 generations, against different P-values. Dataset: SIM4

Each p-value setting was repeated for 25 runs. The standard deviations of the values were small enough to consider the average as meaningful. Value of p was varied from 1% to 95% with more finer-grained coverage in the initial part of the spectrum.

The plot shows no significant trend in the performance for values of p up to about 50%. Subsequently, the quality of the solution deteriorates significantly. Thus, contrary to our expectation, there was no significant loss of performance for small values of p. Plotting a trendline (using a third degree polynomial) shows some variations; but these do not seem to be significant enough for detailed investigation with this dataset.

We performed similar studies with a few other datasets as well. First we show the results with SIM1 dataset with a population size of 20 (See Figure 8.11).

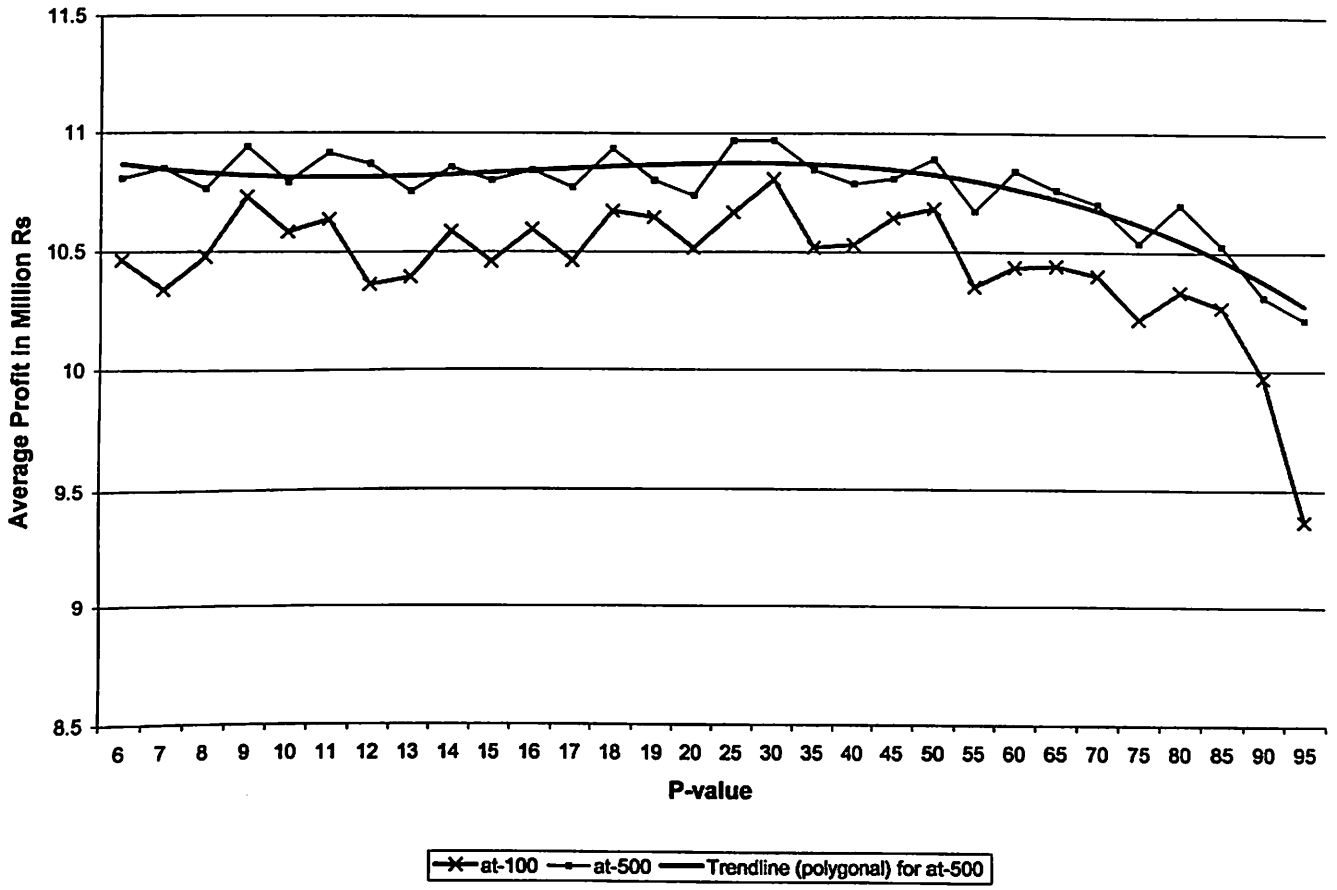


Figure 8.11 Average profit (after 100 and 500 generations) against p-value for dataset SIM1. Retention strategy: s1, Crossover: disabled.

We also tried out this dataset with a different population size (pop=100). As population size increases, the expected quality of solution increases (as was observed in the previous section). Apart from that, there was no major difference from the graph above.

The last graph (Figure 8.12) we present in this thread is for the full dataset *'ACT2'*. This one clearly shows the behaviour we had hypothesized – there is a very clear improvement in solution quality for medium values of p, and visible degradation as we move to both smaller and larger values of p. The degradation is more pronounced for values of p close to 100%.

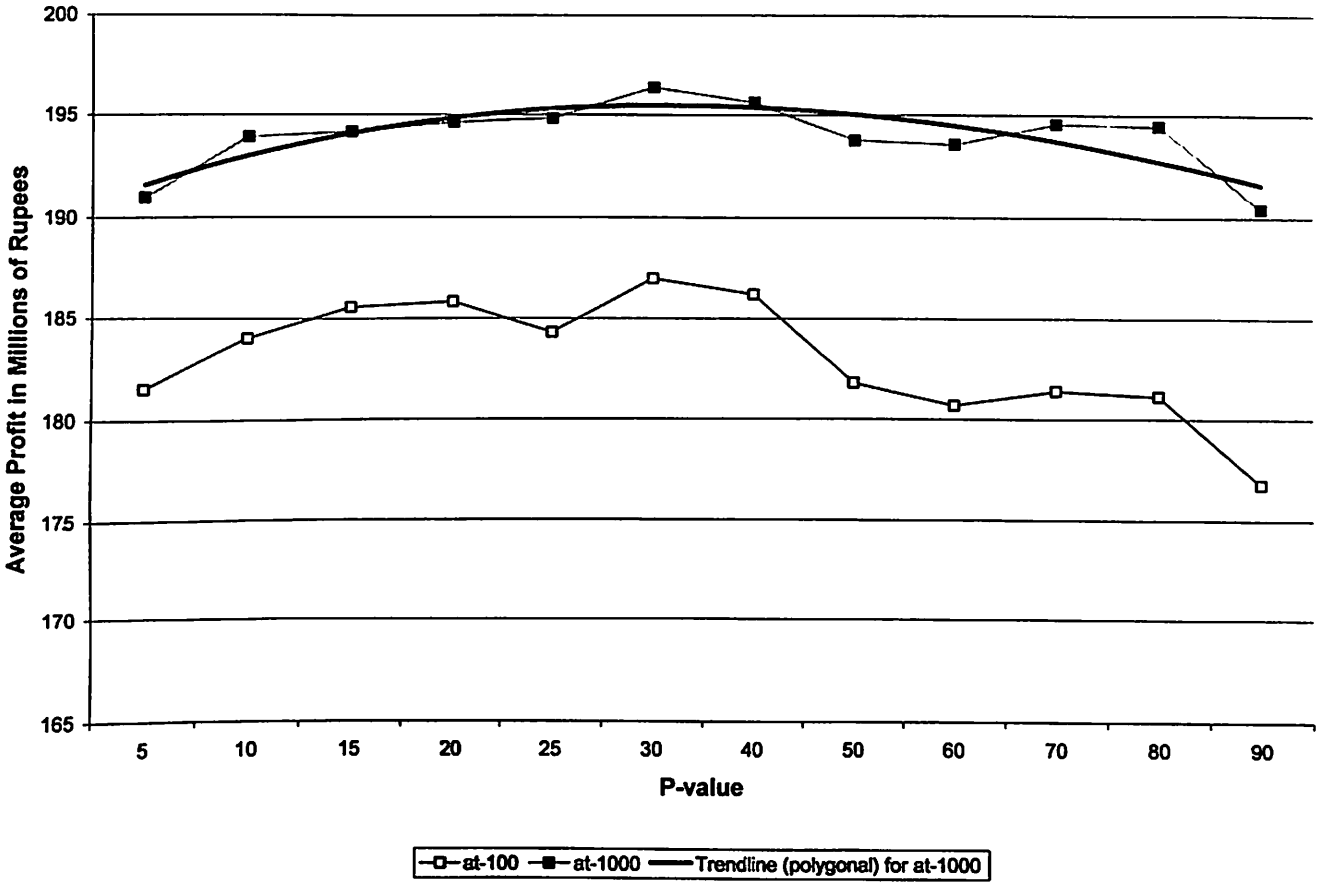


Figure 8.12 Average Profit against P-value (after 100 and 1000 generations) for dataset ACT2.

The experiments seem to indicate a choice of p-value around 20-30% to be best suited. This range is found to be good for all the three sets examined so far.

Why did the STP datasets not show the expected degradation in solution quality as p values decrease towards zero? Further investigations were carried out to answer this question. We tried more variations on one of the STP datasets, namely 'SIM1'. We tried a different retention strategy (see next section) and different operator combination. These two graphs are next shown (See Figure 8.13 and Figure 8.14).

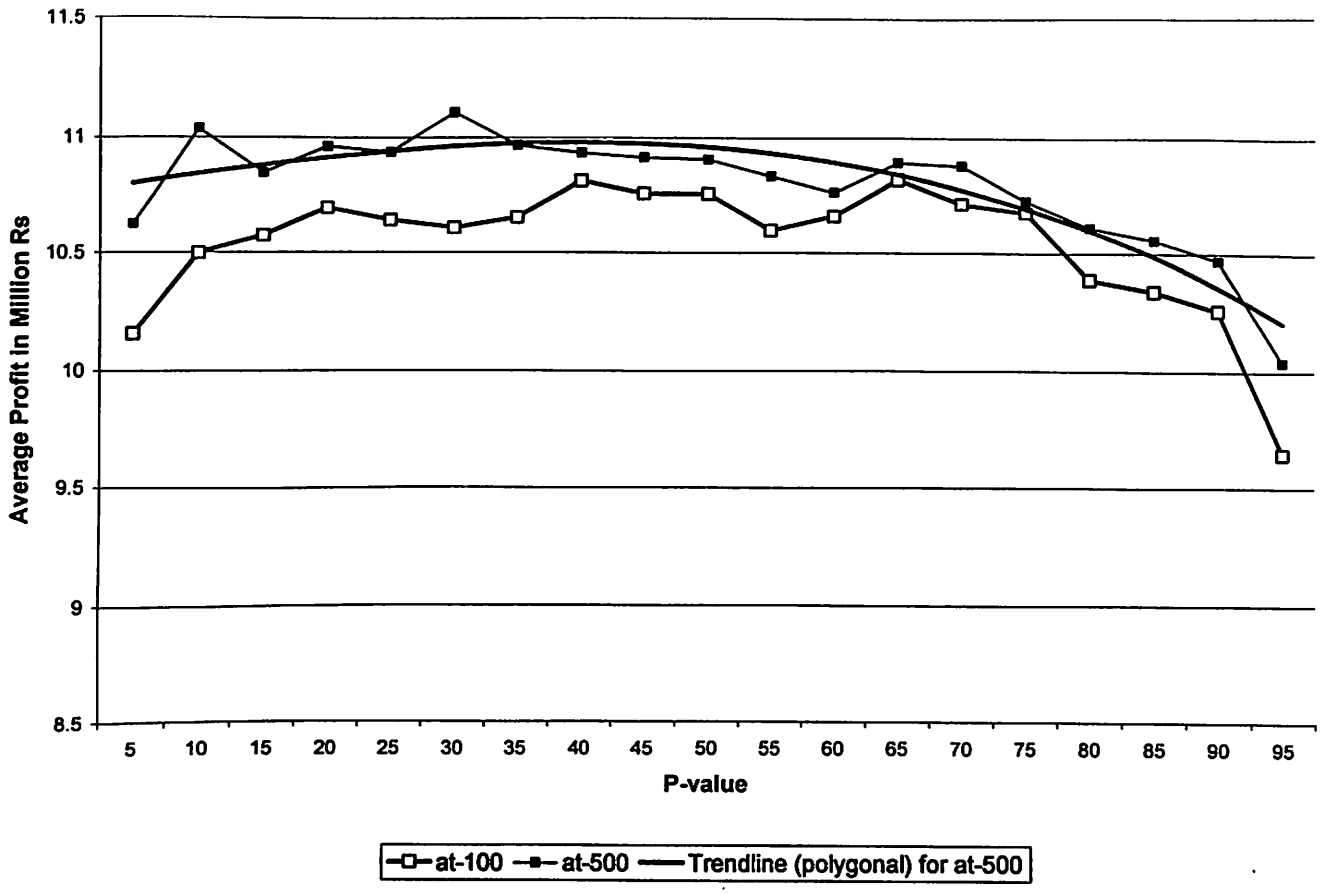


Figure 8.13 Average profit against P-value for dataset SIM1 with retention strategy: s3 and crossover enabled.

These graphs show that the expected behaviour does show up in some cases. For example, when using crossover and mutation, with strategy s3, the behaviour is clearly visible.

In some of the previous experiments also, we have seen the behaviour varying depending on the configuration or the dataset used. As observed by Fogel [Fogel, 2000], there is significant variation on how a particular parameter affects the performance depending on the nature of the dataset. A dataset with difficult-to-find peak or richer constraints may produce different kinds of behaviour against a parameter variation than a dataset without such characteristics.

Figure 8.14 Average profit (after 100 and 500 generations) against P-value for dataset SIM1 under retention strategy s3, crossover disabled.

8.6 Impact of Retention Strategy

The five strategies listed in Section 7.3 were implemented for various datasets and the average profit was measured. Figure 8.15 shows results of one study comparing strategies s1, s3 and s5 for the dataset 'SIM1' against various values of the persistence factor p. As mentioned earlier, p denotes the fraction of schedules retained from one generation to the next.

The graph clearly shows s3 to be the best for most values of p, for this configuration. For very low and very high p-values, s1 shows a better behavior. Since we have already observed that desirable p-value range is 20-40 percent, we need not be concerned with the behaviour for extreme values of p.

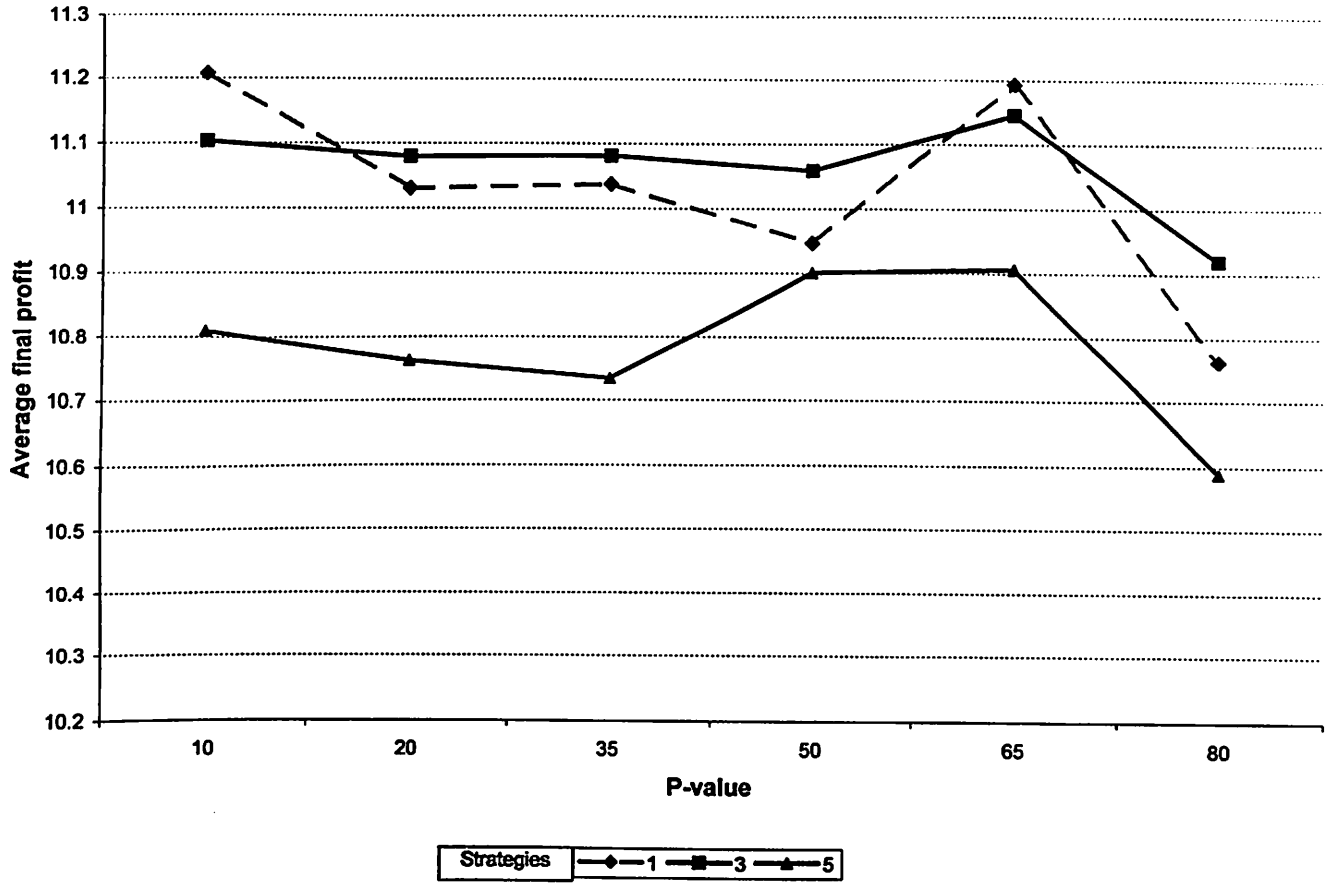


Figure 8.15 Quality of solution against P-value for different retention strategies for dataset SIM1 with population size 50

S5 consistently performs worse than all the strategies. Introducing new solutions during a run contributes negatively to the progress, with the system taking longer to reach the target! This is to be expected in a sense, since a random solution generated at any time is likely to be normally distributed around the overall average. But, as we observed earlier, the average profit of the population steadily increases with the generation count. Thus except during the initial generations, the new solutions introduced are likely to be of much poorer quality compared to the current population average, resulting in immediate rejection. The only impact is one of reducing the effective population size!

Next, we present the results (Figure 8.16) for a different dataset, SIM4.

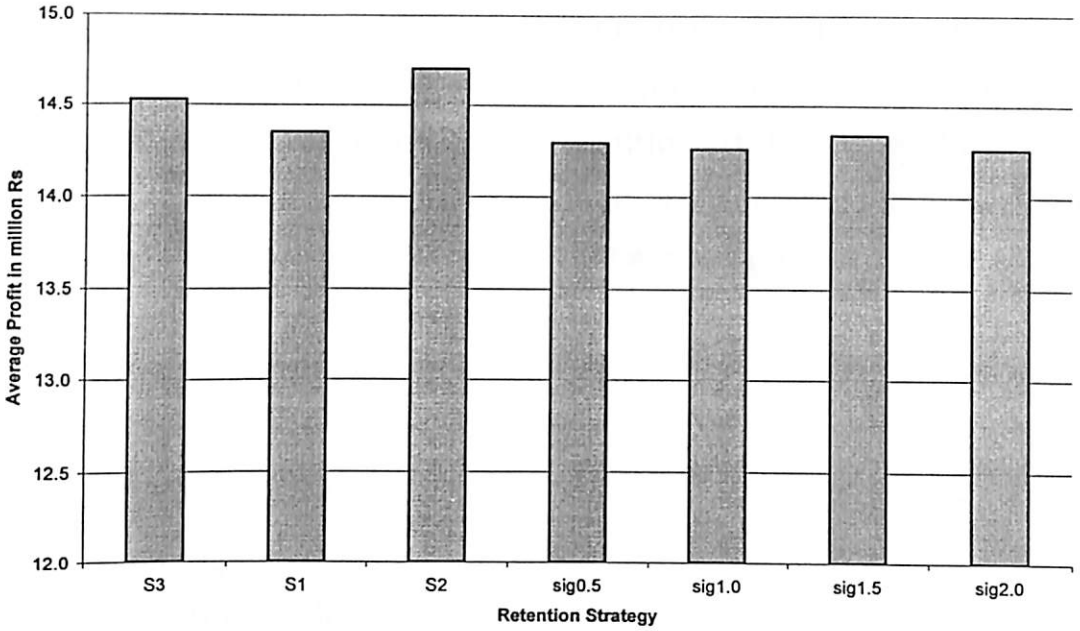


Figure 8.16 Average final cost for various retention strategies with dataset SIM4.

This shows that the difference between s2 and s3 is not very significant, in general. The other observations made earlier are consistent with this study also. In this study, we have included the strategy s4 also. Please recall that strategy s4 discards solutions based on the mean and standard deviation of the current solution set. That is, in each iteration, solutions with profit < μ-kσ are removed from the population. "k" is a constant in a run. We tried different values of k from 0.2 to 2.0. This is shown in the graph as sigxxx where xxx denotes the k value. Larger k indicates that less number of solutions are discarded. We notice that for all values of k tried, the solution quality is poorer than what is obtained with S2 or S3. The next dataset (Figure 8.17) shows this much more clearly, substantiating the observation. All other observations hold for this dataset as well.

Strategies s2 and s3 are clearly superior strategies among those considered. Graphs in Figure 8.11 and Figure 8.14 shown earlier also provide a comparative study of the retention strategies, and report consistent results. Note that the average profit reached in Figure 8.14, which used strategy s3, is better than the average profit reached in Figure 8.11, which used strategy s1.

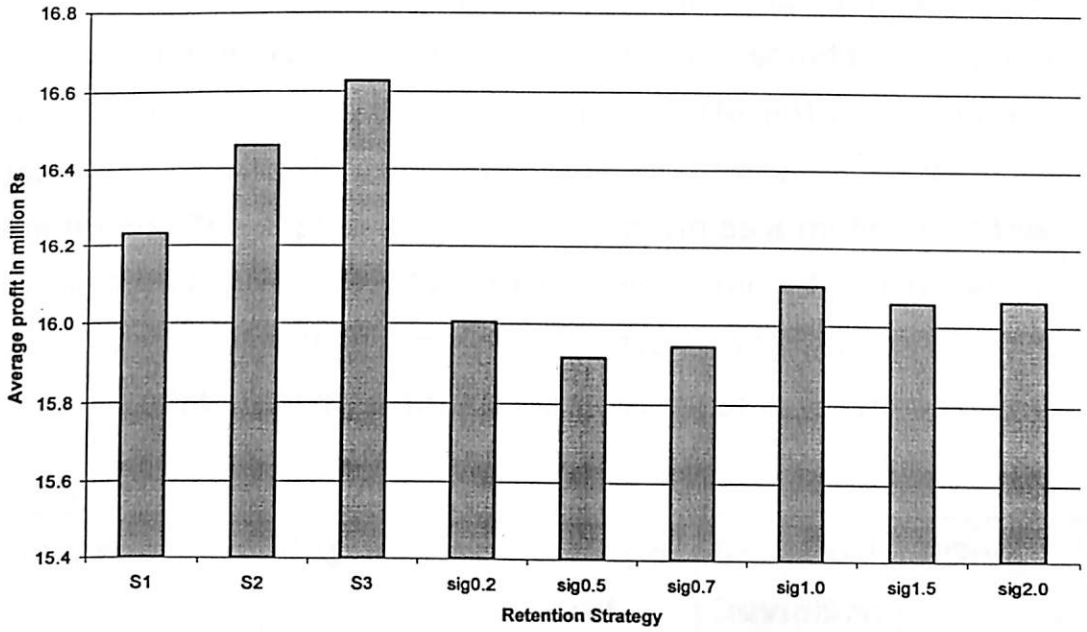


Figure 8.17 Average final profit against various retention strategies for dataset SIM3.

The studies show the following:

- Retention strategies allowing some poor solutions to stay on perform better on average, compared to those retaining only the best.
- Statistical techniques based on mean and standard deviation are not effective. This is partly due to the fact that poor solutions are not given any chance at all. These techniques also suffer from problems such as biasing of the average by extreme solutions.

- Genetic diversity cannot be effectively created, by inserting new random solutions in the population, once the initial iterations are passed.

8.6.1 Handling Duplicate Solutions in the Population

Some trials were carried out for measuring the effect of removing and retaining duplicate solutions in a population; the issues affecting this have been discussed in Section 7.3.1 . Table 8.2 summarises the study showing the measurements of average profit, standard deviation of profit, and average number of generations (NGen) required for various datasets. Final profit is averaged over 25 runs under identical conditions. The standard deviation is shown as a measure of the variability of the final profit across different runs. In each run, we observe the number of generations, beyond which the profit showed no improvement. This generation count averaged is shown as NGen.

Dataset	Duplicates?	Average Profit	Standard Deviation	NGen
SIM2; popsize=100	Retain	12.64	0.35	174
	Remove	12.93	0.20	194
SIM2; popsize=200	Retain	12.87	0.21	220
	Remove	12.99	0.19	227
SIM1; popsize=100 (no crossover)	Retain	11.25	0.18	297
	Remove	11.53	0.11	387
SIM1; popsize=100; crossover.	Retain	11.29	0.21	437
	Remove	11.59	0.09	468

Table 8.2: Comparative Study of Removing and Retaining Duplicates

These show that removing duplicates produces significant improvements in the solution quality. In all the cases tried, the standard deviation shows a tendency to drop. The average number of generations required, NGen, showed no identifiable trend. We can view duplicate solution removal as equivalent to an effective increase in population size – the resulting observations are consistent with this model.

Checking for duplication at the solution-cost level is very cheap to perform. Since population can be maintained as sorted from iteration to iteration, the duplicate checking is at most O(POPSIZE) complexity – which can be further optimized to O(log POPSIZE) if required. And the studies show that the payoff is worth the cost in terms of the probability of potentially different solutions being discarded.

8.7 Generation Count Versus Population Size

The question we raised in this context is the relation between using larger population size and running the system for a larger number of generations. As observed in section 8.2, the solutions have been observed to converge to local optima very early in the run. In addition, we have let some of the runs go over 10,000 generations, and a similar behaviour was observed. The best solution remained unchanged after about 500^{th} generation all the way till 10,000 in most cases. Thus, it is obvious that using larger number of generations does not help significantly to improve the final solution quality obtained.

However, we have seen in section 8.4, that with increasing population size, there is a distinct improvement in the solution quality. Therefore, empirically we see that increasing the generation count does not compensate for reduced population size – which is what we hypothesized in Chapter 7. Figure 8.6 and Figure 8.9 illustrate this point clearly – one can observe that larger population size reaches better profit value than what is possible to reach by increasing the

number of iterations from 100 to 1000. Considering the earlier observation of early convergence in most runs, beyond 1000 iterations, one hardly observes any improvement in solution quality for this dataset.

The following table shows the average of final profit reached at the end of 500, 1000, 2000, and 3000 generations for various population sizes for the dataset SIM2. The average is taken over 20 runs in each case. The number in the bracket in each cell is an estimate (in thousands) of the number of schedules explored during a run of that configuration. This is obtained as the product of number of generations and population size.

Popsize/Generations	**500**	**1000**	**2000**	**3000**
10	11.6 (5)	11.9 (10)	12.0(20)	12.23(30)
50	12.2(25)	12.4(50)	12.6(100)	12.63(150)
100	12.6(50)	12.73(100)	12.8(200)	12.86(300)
200	12.84(100)	12.93(200)	12.98(400)	13.03(600)

This table shows increase in profit value with both increasing generations and increasing population size. However, the profit increase is not purely due to the increased number of schedules being explored as can be seen with the three cells corresponding to exploration of 100 thousand schedules. They correspond to configurations (popsize x number of generations) of 50x2000, 100x1000, and 200x500. The profit value increases within this group as population size increases, despite a decrease in the number of generations. This table, thus, reinforces our observation that returns

from a larger population size is more than the returns from a proportional increase in the number of generations.

8.8 A Consolidated Study

In this section, we take up a new different dataset belonging to the class of STP and review the performance varying all the significant parameters we have observed so far. This dataset named "SIM2" has, apart from reduced demand figures, has temporally non-uniform demands across all city-pairs. For example, all load in Alpha-Gamma segment is in the morning; Delta-Gamma is in the evening; and so on. Sector Alpha-Gamma has very low revenue. The idea is to explore the evolution of time tables that recognize these patterns. Please note that the program used here does not allow any kind of manual direction of the time table based on any of these constraints. Initial time tables are still generated randomly and explored using the mechanisms already outlined.

The settings tried out were:

1. Retention strategies s1 and s3.
2. Population sizes 50, 100, 200, and 500 for both these strategies.
3. Perturbation operators a: only mutation, b: 50% mutation and 50% crossover, c: 80% mutation and 20% crossover, d: only mutation without insert_flight

We observed results consistent with the observations made so far. For example:

- The solution quality was found to increase with increasing population size for both the retention strategies s1 and s3.
- 50% mutation performs worse than 80% mutation.
- When insert_flight is disabled, the solution quality degrades and the number of generations required substantially increases.

- We also observe that, as remarked earlier, for large population size, it is not necessary to have any specialized retention strategy to provide exploration (When population is 500, the best strategy seems to be s1, where as for other values of population size s3 does better).

We also examined some of the best solutions generated by the various configurations. The best one had the profit of 13.4 million Rs. More than one run had produced time tables with this same cost. On analysis, it was found that all these solutions are identical and had the following sector-wise distribution of flights:

Beta-Delta 6 (i.e, 6 flights from Beta to Delta)
Delta-Beta 6
Delta-Alpha 1
Alpha-Delta 2
Delta-Gamma 2
Gamma-Delta 2

The next best set of solutions had a profit of 13.38 million Rs and had the following breakup.

Beta-Delta 5
Delta-Beta 5
Delta-Alpha 1
Alpha-Delta 2
Delta-Gamma 2
Gamma-Delta 2

Beta-Alpha 2
Alpha-Beta 2

These operated the relatively less populated Alpha-Beta segment instead of adding an extra flight to the Beta-Delta sector. Note that converting this to the best solution would require deletion of 4 flights and addition of two. What was more involved was that the two Beta-

Alpha flights were spread out in time (and not consecutive). Thus a series of mutations will be required to transform this solution to the one with profit of 13.4 million Rs, very likely the intermediate solutions will be less attractive than both these solutions. These prevent the current mutation operator from reaching to the 13.4 million Rs state from the current local optima.

8.9 Scheduling for "Cooptition"

As mentioned in Section 7.5, getting airlines to coordinate among themselves to work for an integrated schedule is much more than a case of running the scheduling system on the pooled data. However, we tried out some experiments to study the effect of such pooling. The study was done with dataset SIM1. Consider two airlines A1 and A2, operating a common set of routes with identical privileges and facilities. Consider two scenarios. In one, both A1 and A2 will proceed to make their own time tables independently. For this scenario, we assume that they estimate their load table to be half of the net load available. Let the load table of dataset SIM1 represent this individual estimate. We took one set of runs with the dataset as it is, representing this scenario.

The second scenario is where A1 and A2 decide to pool their resources and work on a cooperation model. Together they target the entire load. So we doubled each of the passenger estimates in the demand table of the SIM1 dataset. To represent pooling of resources, we also doubled the number of aircrafts available for scheduling (ie, sum of aircrafts of A1 and of A2). All other data were left unaffected. The observations are summarized below.

For SIM1, the original run gave an average profit of 11.6 million Rs, with a population size of 100 and number of generations of 2000. Thus the net profit by both A1 and A2 would be 23.2 million Rs. The coordinated schedule gave an average profit of 27.9 million Rs, with same run configuration. This is nearly 20% more than the total profit in

the first scenario. This illustrates the technical feasibility of the cooptition model.

Chapter 9. Use of Parallelism for Performance

Perturbation based solution strategies are very compute-intensive requiring a number of iterations of generations with a fairly large population of solutions. Single machines can generally afford neither the memory nor the computation power to handle realistic values for these parameters. So it is natural to turn to parallelism as a possible approach.

Perturbation model, by its nature, contains relatively few and well-defined points of interaction, and are, therefore, amenable to exploit a parallel or distributed architecture.

9.1 Parallel GA in the Literature

There has been a lot of work in exploiting parallelism in genetic algorithm implementations. [Cantu-Paz, 1995] is a good survey of the approaches and provides a comparison of the various approaches reported. The major approaches [Schoenauer and Michaelewiz, 1997] include:

a) Simple master-slave model where the GA per se is handled by the master, and the evaluation of solutions is done in parallel on other computing units. This is attractive when evaluation of solutions is a complex process which is often the case in practical GA models. In our case also, making a random time table or doing a perturbation on it is much faster than estimating the expected load on all the flights in a time table against the given load table. In many practical problems evaluation of a solution may involve running a time consuming process such as simulation.

b) Massively distributed model where each computing unit hosts a part of the population with restricted crossover scheme (e.g., neighbours only), and

c) Island model which is similar to (b) but individuals migrate to other islands periodically.

In cases of options (b) and (c), there are two further categories: synchronous and asynchronous. If the generations on the different processors proceed with full synchronization (all processors are guaranteed to be in the same generation at any point in time), then the model is synchronous. This may however, demand substantial communication overhead and load balancing efforts, to ensure that the slowest machine do not slow down the whole system. Asynchronous models do not suffer from this; however, the overall view of the system and performance analysis is now more complex.

Given that the sequential GA itself is relatively little understood computationally, the understanding of the effectiveness of the various options in parallel/distributed GA is even less. A number of questions [Schoenauer and Michaelewiz, 1997] are still open:

a. How do the parallel GA differ from sequential GA?

b. What are the expected speed improvements?

c. What are the expected improvements in the quality of the solution?

d. What are the theoretical models underlying parallel GA?

The master-slave approach is most similar to sequential (panmictic) GA, in that, apart from the evaluations of the individuals happening concurrently, all other frameworks of a standard GA will continue to apply. Therefore, one expects the analytical behaviour of sequential GA to carry forward to these. For the other two models, there are changes in the structure. Unlike sequential GA, the parent selection is restricted

to a subset of the total population. Therefore, convergence results, etc that has been done for sequential GA may not apply any more.

The different approaches have reported mixed results in terms of the overall gains – thus leaving the field still quite open.

9.2 Parallelism in Perturbation Models

Our solution approach differs from the GA models in a significant way – the lack of emphasis on crossover. The exploitation of parallelism has largely focused on using crossover to remove complete isolation of the different sub-populations on the different processors. Mutation does not exploit or benefit from the presence of other individuals in a population – except for the pressure of Darwinian selection.

Therefore, the models (and also the constraints) of GA explored for parallelism is unlikely to be very relevant in our model.

There are a few directions, which may provide opportunities to exploit parallelism for perturbation models. Note that our studies so far show that larger population size leads to better quality solutions (on the average). However, single machines may have neither the memory nor the CPU power to process large population sizes. And larger population size means larger running time as well. So the primary focus is to see if the population can be divided across different processors so that effectively a large population size can be achieved, with each processor handling a relatively small population size.

9.3 Our Approach

We have implemented the island model with a central controller whose job is to monitor the progress of the different islands and exchange members if needed.

Each island registers itself with the master on startup, and informs it of the best solution it has got after every generation. The generations among the islands are not synchronized. The master maintains the best

few solutions it has received so far from all the islands put together. When an island consistently reports poor solutions, the master pulls out a random solution from this central pool and asks the island to incorporate it in its pool. This would be better than all the solutions present in the island and therefore, will force exploration from this seed primarily. It is also possible to restart such islands lagging very much behind the others, knowing that we already have better solutions.

9.4 Observations and Analysis

For the purpose of analysis, we simulated parallelism using a multi-island model in the synchronous mode on a single processor machine, with the system cycling through the various islands one by one, in each generation. There is no exchange of solutions among the different islands. Thus the different islands evolve their own population independently. The best solutions are recorded in every generation. The results for two different datasets, SIM1 and SIM2, both from the simplified airline time tabling problem class, are given in Table 9.1.

The table shows the size of population and number of islands used for the various studies. For each combination, 20 runs were taken. The entries in the other columns are averages across 20 runs. The average and standard deviation of the final profit at the end of 1000 generations and the number of generations taken to reach the final profit figure have been tabulated.

		Dataset: SIM1			Dataset: SIM2		
Population size	Number of islands	Average at 1000 generations	Standard Deviation	Number of generations	Average profit at 1000 generations	Standard Deviation	Number of generations
10	20	11549	106	497	13140	165	361
20	10	11556	142	451	13127	97	560
40	5	11430	220	385	12920	228	447
50	4	11492	197	322	13015	193	439
100	2	11403	178	339	13116	201	323
200	1	11419	235	187	12907	222	222

Table 9.1: Comparative Performance of Various Island Distributions

The expression Popsize*Islandcount has been kept as a constant here (at value = 200). The average profit found is quite similar among the different configurations, showing marginally better results for more distributed cases. This is good news for the following reason. Recall that as population size increases, the average solution quality shows a monotonically increasing trend. On the other hand, large population sizes are hard to sustain on a single machine.

What the numbers here shows is that you can take a large population size, and safely distribute that population across multiple machines, exploiting as many machines as available. There is relatively little communication or dependency, in this simple model of distributed computing. At every iteration, the best solution of each island is communicated to the user process (coordinating process) for record. This offers a significant way to exploit parallel processing (even more,

distributed computing on a cluster) in a meaningful way – where parallelism allows us to use a larger population size, which we have seen to improve the solution quality.

This study is not meant as an exhaustive analysis of parallelism models – but to explore the possibility of effective usage of parallelism. We have formulated a simple model that is of much value for effective usage of perturbation based models.

Chapter 10. Conclusion

Perturbation model is indeed quite effective in solving problems such as the ATP effectively. The ability to provide an anytime solution with the promise of more improvement with more time, the ability to start with existing schedules, and the ability to incorporate essentially any kind of constraints in the domain are some of the major advantages in utilizing an iterative improvement approach such as this. In the literature, in general, a trend towards using such iterative improvement methods – of which perturbation model is one example – is visible for complex problems such as scheduling; our studies and observations reinforce that view. In addition, we have the advantage of using the power of distributed computing environments very easily without having to substantially rework the system, or invent specialized algorithms for parallelism. Perturbation based approaches are inherently parallel and well suited to cluster computing kind of environment.

As part of this work, we carried out extensive analysis of the ATP problem and the perturbation model using empirical models as well as formal methods. We presented a proof for NP-completeness of the ATP showing formally the complexity of the problem. This problem, thus, gets added to the computationally challenging list of scheduling problems. We have not seen any work in the literature on the topic of airline time tabling, though a lot of work has been done in related areas such as crew scheduling and gate assignment. Similarly, there are a number of studies being done on classroom and examination time tabling. Airline time tabling is a much more complex problem than these, primarily due to the fact that the number of flights and their durations can have very wide variations. For example, a Mumbai-London-Newyork flight takes nearly 18 hours, where as a Mumbai-Gulf flight takes hardly 3-4 hours. Thus, the number of flights completed by different aircrafts during the given time period, can vary significantly.

There are also major differences between profitability of different flights; the link between percentage occupancy of seats and profitability of flights is non-linear.

This makes it difficult to adapt any of the models proposed for classroom or examination time tabling problems. Given that there is hardly any research work reported in the literature on this particular problem, this thesis is a useful contribution to the field of scheduling – offering another complex problem for further studies.

Empirically we surveyed the search space for a problem of this nature and hypothesised certain properties based on this. Our finding that profit of solutions appear to be normally distributed provided a base to explain some of the observations made during the studies. We believe this observation has much further significance – another topic for possible further investigation. There is no mention of this type of analysis in the literature, to the best of our knowledge.

Our implementation handles a very realistic version of the problem for our study. Based on this implementation, we reported a number of results in terms of the suitability of the approach and the related issues. The focus of the thesis has been to gain insights into this problem and the perturbation model as a viable solution approach to such complex problems.

10.1 Processing Power Versus Quality of Schedule

A 2000-generation run of a STP[13] data with a population size of 100 takes about 30 min on a Pentium III 800 MHz system, but a similar configuration for the ACT1 and ACT2 datasets takes about 15 hours.

We have observed that use of a sufficiently large population size is important, and that population size cannot be compensated by a

---

[13] Simplified Airline Time tabling Problem described in Section 5.1

proportional increase in number of generations. While larger and larger population sizes do not yield proportional increase in quality of schedules produced, we have noticed that quantified utility of the solution obtained does improve with population size (see Figure 8.6, Figure 8.8 and Figure 8.9), whereas the increase in quality with larger generation count is much slower, with larger plateaus of local maxima. For example, in a 20000 generation run of ACT1 dataset, the best solution (profit in million Rs) after each set of 500 generations followed the sequence: 161, 161, 162, 162, 162, 167, 168, 168, 168, 168, 168, 168, 168, 169, 169, 169, 169, 170, 170 showing slow growth. This run took about 3 days of run-time!

In practice, an ATP-like dataset would use a 500 element population with about a 10000 generation run. With the hardware configuration that we used, this would take 3-5 days. With implementation optimizations such as pre-compilation and better data structures this time may reduce by 50%. As mentioned in Section 1.2, for a large airline, the data can be 10-100 times larger than ATP and hence we are facing fairly large run time in real-life scenarios. For optimal use of computer aided scheduling technology, it would be valuable to obtain a response time of 10-15 minutes. This would enable the scheduling experts to experiment with various combinations and alternatives, before choosing a schedule.

Profit improvements of over 5-10% have been observed with use of larger population sizes and 2-5% by using large number of generations. These could translate to a few billions of Rupees per year for even a modest international airline. High performance multiprocessor machines today cost about a million rupees – a worthwhile investment for almost any airline, based on the above observations. High performance cluster computing environments are also becoming highly available. These are "software based supercomputers" exploiting a distributed memory architecture and standard COTS desktop and workstation computers.

These would cost typically an order of magnitude less than specialised supercomputers and hence tilt the balance even more in favour of investing in a high performance computing environment.

Given the lack of vector computing operations in perturbation model, a conventional super computer such as CRAY built over high-speed vector computing philosophy is not likely to be a worthwhile investment. The costs are also likely to be higher in comparison to the cluster computing machines mentioned above.

10.2 Summary of Major Results

The major observations based on the analysis and studies carried out in this thesis, can be summarized as follows.

a) Perturbation based approach to solving complex scheduling problems is viable and practical. It offers significant advantages over other competing models to be given serious consideration as the practical solution.

b) The search space of these problems is moderately vicious. Our sampling analysis reported in Chapter 5 and the other studies, increase our confidence in this hypothesis. The sampling analysis of the search space is unique, as far as we know.

c) The population size is an important factor determining the run-time and the quality of solution. The value should not be too small or too large. We have found that a population size of a few hundreds is ideal for the problem sizes studied. Larger values would be required for constraint rich and large datasets. It should be selected based on preliminary studies. We also observe that running a smaller population for a larger number of iterations is much less effective than a larger population running for a smaller number of generations.

d) Population management consisting of determining persistence factor, duplicate removal and retention strategy also influences the quality of

the solution arrived at in a given time. Retention strategies, which exploit available good solutions as well as encourage some amount of random exploration, are found to do better. Strategies s2 and s3 do better in most cases compared to the other strategies tried. Persistence factor shows the best behaviour – in terms of quality of solution obtained - for values in the range of 20-50 percent.

e) Despite many studies to the contrary, crossover is seen not to play a critical role in perturbation-based models of this nature. Mutation is alone adequate as the operator. Crossover, if present, can be used to speed up convergence.

f) Largely, perturbation based approaches are not critically dependent on specific problem parameters. One can easily add or delete additional parameters or constraints without having to review the model. The tolerance to variations in parameter values is quite good for practical usage. The selection pressure imposed by the evaluation function seems to be the primary driving force in perturbation model.

g) Operators should be carefully chosen keeping in mind their ability to guarantee wide enough reach in the solution space, low probability of generating invalid solutions, and ease of implementation. Depending on the nature of datasets, small differences in the operator functionality can make significant improvements in performance. We believe that probabilistic reachability – the probability of reaching a given state from any other state, within a finite number of iterations – is an important factor for investigation, and may provide valuable insights into operator selection for iterative improvement models. It also seems to be the case that if this property can be assured, it may not be necessary to invent too many specialized operators for specific problems. Note that we have been able to obtain good results, with the simple mutation operator for all the datasets we tried.

h) We proposed a benchmark problem (STP) in this category and a number of datasets (SIM1, SIM2, SIM3 and SIM4) for that problem, for facilitating further research in this area.

i) Our experiments with parallelism have demonstrated that a cost effective approach to utilize higher computing power to get better quality solutions, by expanding the population size, is very practical. This approach can be effectively used with low cost solutions such as cluster computing, since the coordination and synchronization overheads are very small. This is unlike the other parallel computing models – such as using vector computers -, where the payoff is dependent on a number of parameters such as memory bandwidth and is bounded by the hardware constraints of the setup. Cluster computing costs very little to develop and deploy, and shows high degree of scalability.

j) We have proposed and investigated the problem of cooptitive scheduling where a number of airlines pool their resources together against a combined load model to prepare an integrated time table and then distribute the flights among themselves for actual operation. This is a strategically important problem, where practical viability has been demonstrated by our work.

k) We have built up a framework to characterize perturbation type approaches through a set of parameters including population size, persistence factor, retention strategy, crossover-mutation ratio, etc. The formulation of the concepts of retention strategy and persistence factor is unique and novel as far as we know. We also present a serious and systematic study of their impact on the solution quality.

10.3 Future Work

Perturbation models are being accepted as a reliable and effective solution method for solving complex optimization problems including real-life scheduling problems. However, almost every particular

application of this model for solving a problem is being hand-crafted with substantial tuning and refinement for achieving good performance. The inherently stochastic nature of the model and the fact that a number of aspects are left open to the implementer, have made this field a predominantly empirical one. Despite the substantial amount of work done using Genetic algorithms and general iterative refinement models, practical guidelines for effective exploitation of perturbation approach are still evolving. This thesis has explored a particular problem – a practically significant and computationally complex one – for study using the perturbation approach. A number of interesting results have been obtained, many of which have significance in dealing with other problems of this class.

At the same time, we have identified a number of further questions and issues emerge out of the study. These would help us develop further the empirical analysis of the perturbation model, along the lines proposed by Hooker [Hooker, 1994]. Such an attempt should probe further the factors of the problem domain that control the convergence rate, optimal population size to be used, choice of retention strategy, etc.

We have explored the exploitation of parallel environment for perturbation models. With the substantial excitement over cluster and grid computing [Foster and Kesselman, 1999] [Sterling, 2001], massive computational power can be made available without much extra cost for solving such computationally complex problems. The model that we have explored suffers comparatively little communication overhead, and performs quite well. Experimenting with massive computing power through grid and cluster computing to solve problems of this type would be valuable.

In the previous section itself, we have highlighted some of the areas of further work – to better exploit some of the observations we had during this study and to better understand their significance.

Appendix A: Implementation Details

The major classes in the implementation and their functionality are as follows. Each major entity (aircraft, for example) and group of entities (vehicleschedule, for example) in the problem domain has been implemented as a class, which provides methods to support their relevant functionality.

Class-name	**Short description**
Client.java	Invocation class for the slave. Manages interaction with the master, and starts up Main.
Config.java	Keeps track of configuration information (from the config file) such as mode of initialization, number of iterations, etc
DemandType.java	Represents one entry of demand model, and supports associated functionalities.
DistMatrix.java	Handles the inter-location distances
Event.java	An event is one non-stop flight from one airport to another. An instance of this class represents an event.
Filebuf.java	Used for charge-data files. The class loads and keeps the file contents internally and computes charges as per the defined format.
Flight.java	This class represents a flight, which is a sequence of events corresponding to a given route.
Global.java	Keeps track of data and methods which need to be accessed by many different parts of the

	system: vehicle-list, location-list, random number generator, etc.
Infile.java	Class for reading text files
Item.java	Represents one type of passenger. One instance for each type.
ItemListType.java	List of passenger types
ItemScenario.java	Represents a demand scenario, associated with a given time table. Structurally similar to the load table input. After loading the time table, this is updated to reflect the unmet demands.
ItemStore.java	Part of ItemScenario. Keeps demand from a given location.
LoadType.java	Represents one demand entry <from, to, pax type, number, pref time, pref day> as given in the load model.
Location.java	Represents one location
LocationListType.java	List of locations
Log.java	Used for generating log of run. Manages the log file.
Main.java	Main class - invoked by slave.
MainSeq.java	Auxiliary class used for the search space study - generates a given number of random schedule instances.
Outfile.java	File for writing text files.
Perturb.java	Represents a population of N solutions, and is concerned with the management of the

	population.
Roulette.java	Implements Roulette wheel model of selection in a general form.
Route.java	Represents a route
Routes.java	Represents the full set of routes available. Supports retrieving routes as per specified criteria such start-from A, end at B, etc.
Scenario.java	Represents one solution (schedule) along with the load-table.
Schedule.java	Represents a time table as a set of vehicle schedules.
Server.java	The main class for the master component
ServerRecord.java	Class handling recording of information on each slave.
ServerUI.java	Handles the User interface – display and interaction with user – of master.
TimeU.java	Represents a time interval. Handles conversion from minutes to hr:min format and back.
Trace.java	Recording trace of a run.
UI.java	Slave's user interface.
Vehicle.java	Represents a vehicle
VehicleListType.java	Represents the list of vehicles
VehicleSchedule.java	Represents a vehicle schedule, as a set of flights.

Appendix B: Samples of Datasets and Output

B.1 Sample Dataset (SIM1)

Master data

validvehicletypes 5 bo747 ab310 ab300 combi d400

transit_time 30

max_time 1080

load_drop_off_time 120

end

Locations

dummy dum 0 0 *

Beta bom 0 1 *

Delta del 0 1 *

Gamma cal 0 1 *

Alpha mds 0 1 *

Vehicles

ab310:1 ab310 200 854 7520 480 45000

ab300:1 ab300 250 851 5500 480 45000

ab300:2 ab300 250 851 5500 480 45000

Distances

Beta Alpha 90

Beta Gamma 120

Beta Delta 110

Alpha Gamma 150

Alpha Delta 150

Alpha Beta 90

Gamma Delta 120

Gamma Beta 120

Gamma Alpha 150

Delta Beta 110

Delta Alpha 150

Delta Gamma 120

Routes

2 r1 Beta Delta

2 r3 Beta Alpha

2 r4 Beta Gamma

2 r9 Gamma Delta

2 r10 Gamma Beta

2 r12 Gamma Alpha

2 r13 Delta Beta

2 r15 Delta Alpha

2 r16 Delta Gamma

2 r18 Alpha Gamma

2 r19 Alpha Delta

2 r20 Alpha Beta

3 r21 Alpha Beta Delta

3 r22 Delta Beta Alpha

0

Event charges

Beta * * * * 10000 landing

Gamma * * * * 10000 landing

Delta * * * * 10000 landing

Alpha * * * * 10000 landing

Demand table

Alpha Delta eco_pass -1 120 200

Alpha Delta eco_pass -1 840 200

Alpha Gamma eco_pass -1 120 50

Alpha Gamma eco_pass -1 300 100

Alpha Gamma eco_pass -1 600 50

Alpha Gamma eco_pass -1 840 100

Alpha Beta eco_pass -1 120 100

Alpha Beta eco_pass -1 600 100

Delta Alpha eco_pass -1 300 150
Delta Alpha eco_pass -1 840 150
Delta Gamma eco_pass -1 300 150
Delta Gamma eco_pass -1 600 150
Delta Beta eco_pass -1 120 200
Delta Beta eco_pass -1 300 200
Delta Beta eco_pass -1 600 200
Delta Beta eco_pass -1 840 200
Gamma Alpha eco_pass -1 120 50
Gamma Alpha eco_pass -1 300 50
Gamma Alpha eco_pass -1 600 100
Gamma Delta eco_pass -1 120 200
Gamma Delta eco_pass -1 300 100
Gamma Beta eco_pass -1 120 150
Gamma Beta eco_pass -1 600 100
Gamma Beta eco_pass -1 840 100
Beta Alpha eco_pass -1 300 100
Beta Alpha eco_pass -1 600 100
Beta Alpha eco_pass -1 840 100
Beta Delta eco_pass -1 120 500
Beta Delta eco_pass -1 840 500
Beta Gamma eco_pass -1 120 50
Beta Gamma eco_pass -1 600 50
Beta Gamma eco_pass -1 840 50

Link Charges

* * * bo747 time 94000 fuel
* * * bo747 time 94000 general
* * * ab310 time 45000 fuel
* * * ab310 time 45000 general
* * * ab300 time 45000 fuel
* * * ab300 time 45000 general

Revenue

Alpha Delta eco_pass 6500

Alpha Gamma eco_pass 6500

Alpha Beta eco_pass 3000

Delta Alpha eco_pass 6500

Delta Gamma eco_pass 5500

Delta Beta eco_pass 3500

Gamma Alpha eco_pass 6500

Gamma Delta eco_pass 5500

Gamma Beta eco_pass 5000

Beta Alpha eco_pass 3000

Beta Delta eco_pass 3500

Beta Gamma eco_pass 5000

B.2 Sample Schedule Generated

Aircraft: ab310:1

Route	From	time	To	time
r9	Gamma	30	Delta	150
r15	Delta	180	Alpha	330
r18	Alpha	360	Gamma	510
r12	Gamma	540	Alpha	690
r19	Alpha	720	Delta	870
r15	Delta	900	Alpha	1050

Aircraft: ab300:2

Route	From	time	To	time
r1	Beta	90	Delta	200
r16	Delta	230	Gamma	350
r9	Gamma	380	Delta	500
r16	Delta	530	Gamma	650
r10	Gamma	680	Beta	800
r1	Beta	830	Delta	940

Aircraft: ab300:1

Route	From	time	To	Time
r10	Gamma	30	Beta	150
r1	Beta	180	Delta	290
r13	Delta	320	Beta	430
r1	Beta	460	Delta	570
r13	Delta	630	Beta	740
r1	Beta	770	Delta	880
r13	Delta	910	Beta	1020

Glossary

These are specialized terms used in this thesis. The terms are alphabetically ordered for ease of reference.

ATP: The airline time tabling problem in general, as described in Chapter 1.

Average Profit: For each setting of the parameter values, the program is run many times (typically 20) independently. In each run, the best solution reached at the end of the run and the corresponding profit value are recorded. Since perturbation model is stochastic in nature, this profit value may vary from run to run. (See 130 also) Therefore, to cancel the stochastic effects, the average of these best profit values across the different runs are used in our study for analysis and comparison. This is referred to as "average profit" in this thesis.

Benign search space: A smooth search space, where the difference between the quality of a solution and that of its neighbours does not show sudden jumps or falls.

Block-time: non-stop flying time between two airports. Block-times are normally given separately for different types of aircrafts.

Chromosome: (borrowed from the genetic algorithm literature) The data structure used to represent a solution, or a partial solution.

Crossover: change in a schedule created by combining parts of more than one schedule (normally two).

Curfew time (at an airport): During these periods, no aircraft is allowed to land or takeoff at that airport.

Feasible schedule: A schedule that can be implemented technically, that is, no hard constraints are violated.

Generation: One iteration of perturbations over the current population. A new set of candidate schedules are created from the

current set of candidate schedules using available perturbation operators, and a subset of these two sets together is selected to be the population for the next generation.

Hop: A non-stop flight segment.

Mutation: A unit change created in a schedule in an attempt to create a better (alternative) schedule.

Optimal schedule: A schedule that is feasible and has the best profit among all schedules possible with the given data and constraints.

Perturbation: A change made to a given schedule. Normally these are small changes. For an airline time table, such a change can be addition of a flight, removal of a flight, or change of a flight parameter such as route, departure time, aircraft used, etc for a flight.

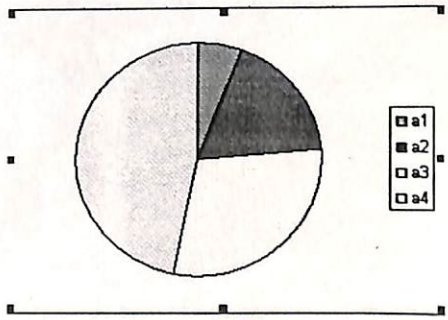
Persistence factor: The fraction of solutions retained from one generation to the next generation, expressed as a percentage.

Population: A set of solutions used for exploration.

Range of an aircraft: The distance that the aircraft can fly non-stop.

Roulette wheel: Derived from the rotating wheel used in gambling, this is a device for implementing weighted random selection from a given set of options. For example, consider 4 individuals a_1, a_2, a_3, a_4, of age 10, 30, 50, and 80 respectively and we want to randomly select one of these, favouring the older ones (i.e. the older you are, the more likely you are of being selected). We can allocate them space on a wheel proportional to the age. Space allocated to a_i will be $age_i / \Sigma_j age_j$ of the 360^0 available.

In the example, this will be 21, 63, 105, and 170, giving an allocation of the wheel as shown. The wheel is now rotated and allowed to rest naturally and the area corresponding to the topmost point is



noted. It can be observed that since nearly half the circle is occupied by a_4, and every point on the circle is equally likely to be at the top at rest, a_4 will get selected nearly 50% of the time.

This model is used in parent selection in GA and perturbation models and wherever a biased (weighted) selection is required.

Search space: The set of all possible schedules.

Sector: A pair of locations covered by a flight, possibly with stop-overs en-route. Consists of one or more hops.

Slot-time at an airport: Normally applicable in very busy airports, any landing or takeoff at such an airport by a specific airline must be at this specified time.

STP: Simplified version of the airline time tabling problem. The simplifications are defined in Section 1 of Chapter 5. Just as the drosophila is a model insect for the biologists, the STP provides us with a simple and yet realistic model for studying airline time tabling problem within reasonable time and resource bounds.

Vicious search space: Search space, in which the difference between quality of two neighbouring solutions can be unpredictably large.

References

[Aarts and Lenstra, 1997] Aarts EHL and Lenstra JK. (eds). *Local Search in Combinatorial Optimization*. Wiley, Chichester, 1997.

[Allen et al, 1990] Allen J, Hendler J and Tate A. *Readings in planning.* Morgan Kaufmann, 1990.

[Arabeyre et al, 1969] Arabeyre J.P., Fearnley J., Steiger F.C., Teather W., *The airline crew scheduling problem: a survey,* Transport. Science, Vol 3, pp 140-163, 1969.

[Baase, 1988] Baase S. *Computer Algorithms: Introduction to Design and Analysis*. Addison-Wesley, 1988.

[Baker, 1974] Baker KR. *Introduction to sequencing and scheduling.* John Wiley and Sons. 1974.

[Beck et al, 1997] Beck JC, Davenport AJ and Fox MS. *Five pitfalls of empirical scheduling research*. Principles and practice of constraint programming, pp 390-404, 1997.

[Bodin, 1990] Bodin LD. *Twenty years of routing and scheduling*. Operations Research, Vol 38(4), July/August 1990.

[Bodin et al, 1983] Bodin LD, Golden B, Assad A and Ball M. *Routing and scheduling of vehicles and crews - the state of the art*, Computers and Operations Research. Vol 10, pp 63-211, 1983.

[Brannlund et al, 1998] Brannlund U, Lindberg PO, Nou A, and Nilsson JE. *Railway timetabling using Lagrangian relaxation.* Transportation science, Vo 32(4), November 1998.

[Brazil and Swigger, 1988] Brazil RP and Swigger MK. *GATES: An airline gate assignment and tracking expert system.* IEEE Expert, vol 3, pp 33-40, summer 1988.

[Burke and Newall, 1999] Burke EK and Newall JP. *A multi-stage evolutionary algorithm for the timetable problem.* IEEE Transactions on Evolutionary Computation, Vol 3(1), pp 1085-1092, 1999.

[Carter, 1986] Carter MW. *A survey of practical applications of examination timetabling algorithms.* Operations Research, Vol 34/2, pp 193-202, March 1986.

[Carter and Laporte, 1996] Carter MW and Laporte G. *Recent developments in practical examination timetabling.* Proceedings of first international conference on "The practice and theory of automated timetabling", pp 3-21, 1996.

[Cantu-paz, 1995] Cantu-paz E. *A summary of research on parallel genetic algorithms.* IlliGAL Report 95007, University of Illinois at Urbana-Champaign, July 1995.

[Caprara et al, 1998] Caprara A, Toth P, Vigo D and Fischetti M. *Modeling and solving the crew rostering problem*. Operations Research, 46(6):820-830, 1998.

[Chiang and Hau, 1995] Chiang Te-Wei and Hau Hai-Yen. *Railway scheduling system using repair based approach*. In Proceedings of the Seventh international conference on Tools with AI, 1995.

[Chu et al, 1995] Chu HD, Gelman E, and Johnson EL. *Solving large scale crew scheduling problems,* European J. Opl. Research, Vol 97, 1997, 260-268

[Cohen, 1990] Cohen J. *Constraing logic programming languages.* Communications of the ACM. Vol 33(7), pp 52-68, July 1990.

[Cordeau et al, 1998] Cordeau Jean Francois, Toth Paolo and Vigo Daniele. *A Survey of optimization models for train routing and scheduling.* Vol 32(4), Transportation Science, 1998.

[Cormen et al, 1990] Cormen TH, Leiserson CE and Rivest RL. *Introduction to Algorithms.* MIT Press, 1990.

[Corne et al, 1999] Corne D, Dorigo M and Glover F. *New Ideas in Optimization.* McGraw-Hill, 1999.

[Dhar and Ranganathan, 1990] Dhar V and Ranganathan N. *Integer programming vs expert systems: an experimental comparison.* Communications of the ACM, March 1990.

[Dorn et al, 1994] Dorn J, Girsch M, Skele G and Slany W. *Comparison of iterative improvement techniques for schedule optimization.* Proceedings of the 13th UK Planning Special Interest Group , Glasgow (also available as CD-TR 94-61), 1994.

[Dorn, 1995] Dorn J. *Iterative improvement methods for knowledge based scheduling.* AI Communications, 8/1, 1995, pp 20-34.

[Duncan, 1995] Duncan Tim. *Schedule-IT: An intelligent vehicle scheduling system.* Technical Report AIAI-TR-180, Artificial Intelligence Applications Institute, University of Edinburgh, 1995.

[Fang, 1994] Fang Hsiao-Lan. *Genetic Algorithms in timetabling and scheduling.* Phd thesis. Department of AI, University of Edinburgh, 1994.

[Fikes and Nilsson, 1971] Fikes RE and Nilsson NJ. *STRIPS: a new approach to the application of theorem proving to artificial intelligence.* Artificial Intelligence, Vol 1(2), 1971.

[Fogel and Stayton, 1994] Fogel DB and Stayton LC. *On the effectiveness of crossover in simulated evolutionary optimization.* Biosystems. Vol 32, pp 171-182, 1994.

[Fogel, 2000] Fogel DB. *Evolutionary computation: toward a new philosophy of machine intelligence.* IEEE Press. 2000.

[Forrest and Mitchell, 1993] Forrest S and Mitchell M. *What makes a problem hard for genetic algorithm? Some anomalous results and their explanation.* Machine learning, Vol 13, pp 285-319, 1993.

[Foster and Kesselman, 1999] Foster I and Kesselman C (Eds). *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufman, San Francisco, 1999.

[Fox, 1987] Fox MS. *Constraint-directed search: a case study of job-shop scheduling.* Morgan Kaufmann, 1987.

[Fox and Smith, 1984] Fox MS and Smith SF. *ISIS: A knowledge-based system for factory scheduling.* Expert systems, Vol 1(1), pp 25-49, 1984.

[Fox, 1994] Fox MS. *ISIS: A retrospective.* In [Zweben and Fox, 1994], pp 3-28.

[Foxley and Lockyer, 1968] Foxley E and Lockyer K. *The construction of examination timetables by computer.* The computer journal. Vol 11, pp 264-268, 1968.

[Frangouli et al, 1995] Frangouli H, Harmandas V and Stamatopoulos P. *UTSE: Construction of optimum timetables for university courses – a CLP based approach.* 3rd International Conference on the Practical Applications of Prolog, Paris, April 1995. citeseer.nj.nec.com/150450.html

[FSAS, 1999] *Flight Schedule Automation System,* http://www.airauto.com/aai/products_fsas.html.

[Georgeff, 1988] Georgeff MP. *Reasoning about plans and actions.* In [Shrobe and AAAI, 1988], pp 173-196.

[Gillet, 1976] Gillet BE. *Introduction to Operations Research: a computer oriented algorithmic approach.* Tata McGraw Hill, 1976.

[Glover, 1990] Glover F. *Tabu search: a tutorial.* Interfaces Vol 20(4), pp 74-94; 1990.

[Goldberg, 1989] Goldberg DE. *Genetic algorithms in search, optimization and machine learning.* Addison-wesley, 1989.

[Goldberg, 1998] Goldberg DE. *A meditation on the application of genetic algorithms.* IlliGAL Report No 98003, Univ of Illinois at Urbana-Champaign, 1998.

[Goldstein and Roberts, 1977] Goldstein IP and Roberts RB. *NUDGE – a knowledge based scheduling program.* AI Memo 405, MIT AI Lab, 1977.

[Grant, 1986] Grant TJ. *Lessons for O.R from A.I: A scheduling case study.* Journal of operations research society, Vol 37(1), pp 41-57, 1986.

[Gray et al, 1997] Gray P, Hart W, Painton L, Phillips C, Trahan M and Wagner J. *A Survey of Global Optimization Methods.* http://www.cs.sandia.gov/opt/survey/main.html, Sandia National Laboratories, Albuquerque, 1997.

[Hentenryck, 1994] Hentenryck P. *Scheduling and packing in the constraint language CC(FD).* In [Zweben and Fox, 1994], pp 137-168.

[Hentenryck, 1996] Hentenryck P. *Constraint programming for combinatorial search problems.* ACM Computing surveys, December 1996.

[Henz and Wurtz, 1995] Henz M and Wurtz J. *Using OZ for college timetabling.* Proceedings of the 1995 International Conference on the Practice and Theory of Automated Timetabling.

[Holland, 1992] Holland J. *Adaptation in natural and artificial systems.* MIT Press, 1992.

[Hooker, 1994] Hooker JN. *Needed: An empirical science of algorithms.* Operations research, Vol 42, Issue 2, pp 201-212, March-April 1994.

[Horstmann, 2000] Horstmann Cay. *Computing concepts with Java 2: essentials (second edition).* John Wiley and Sons, 2000.

[Ignizio, 1983] Ignizio JP. *Generalised goal programming: an overview.* Computers and operations research, Vol 10(4), pp 277-289, 1983.

[Ilog, 1997] ILOG, Inc. *Optimisation technology white-paper: a comparative study of optimization techniques.* 1997

[Jain and Meeran, 1998] Jain S and Meeran S. *A state-of-the-art review of job-shop scheduling techniques.* Technical report, Department of Applied Physics, Electronic and Mechanical Engineering, University of Dundee, Dundee, Scotland, 1998.

[Johnson et al, 1988] Johnson D. S, Papadimitriou C. H and Yannakakis M. *How Easy is Local Search?*, Journal of Computer and System Sciences, 37(1), pp79-100, 1988.

[Jennings et al, 1998] Jennings NR, Sycara KP and Wooldridge M. *A roadmap for agent research and development.* Journal of autonomous agents and multiagent systems. Vol 1(1), pp 7-36, 1998.

[Kolodner, 1993] Kolodner JL. *Case based reasoning.* Morgan Kaufmann, 1993.

[Korf, 1987] Korf RE. *Planning as search.* Artificial Intelligence, pp 65-88, 1987.

[Korf, 1988] Korf RE. *Search: A survey of recent results.* In [Shrobe and AAAI, 1988], pp 197-237.

[Magnanti, 1981] Magnanti TL. *Combinatorial optimization and vehicle fleet planning: perspectives and prospects.* Networks Vol 11, pp 179-213, 1981.

[Mammen and Hogg, 1997] Mammen DL and Hogg T. *A new look at the easy-hard-easy pattern of combinatorial search difficulty.* Journal of Artificial Intelligence Research, Vol 6, pp 47-66, 1997.

[Minton et al, 1990] Minton S, Phillips A, Johnson M and Laird P. *Solving large-scale CSP and scheduling problems with a heuristic repair method.* Proceedings of AAAI-90.

[Mitchell et al, 1992] Mitchell M, Forrest S and Holland J H. *The royal road for genetic algorithms: Fitness landscapes and GA performance.* In

Proceedings of the First European Conference on Artificial Life. Cambridge, MA: MIT Press/Bradford Books. http://citeseer.nj.nec.com/mitchell91royal.htm, 1992.

[Mooney, 1995] Mooney EL. *Local search algorithms*. Technical report of Industrial and management engineering department. Montana State University, Montana. May 1995.

[Nilsson, 1980] Nilsson, NJ. *Principles of Artificial Intelligence.* Morgan Kaufmann, 1980.

[Russel and Norvig, 1994] Russel SJ and Norvig P. *Artificial Intelligence: A modern approach*. Prentice Hall, 1994.

[OASIS, 1999] Oasis, http://www.drba.com/Oasis.html

[Park and Carter, 1995] Park K and Carter B. *On the effectiveness of genetic search in combinatorial optimization*. In Proceedings of the 10th ACM Symposium on Applied Computing. ACM Press, 1995. http://citeseer.nj.nec.com/article/park95effectiveness.html

[Rich and Knight, 1992] Rich E and Knight K. *Artificial Intelligence* (second edn). McGraw Hill, 1992.

[RMI, 1999] *Getting started using RMI,* http://www.javasoft.com/products/jdk/1.1/docs/guide/rmi/getstart.doc .html

[Rushmeier et al, 1995] Rushmeier RA, Hoffman KL, and Padberg M. *Recent advances in exact optimization of airline scheduling problems*. Technical report, George Mason University, July 1995.

[Sasikumar et al, 2000] Sasikumar M, Shikhare DS and Prakash PR. *Introduction to Parallel Processing*. Prentice Hall India, 2000.

[Schoenauer and Michalewicz, 1997] Schoenauer M and Michalewicz Z. *Evolutionary Computation*. Control and Cybernetics, Vol 26, No 3, pp 307-338, 1997.

[Shrobe and AAAI, 1988] Shrobe HE and AAAI (ed). *Exploring Artificial Intelligence – survey talks from the national conferences on artificial intelligence*, Morgan Kaufmann, 1988.

[Shulte et al, 1998] Shulte C, Smolka G, and Wurtz J. *Finite domain constraint programming in Oz: A tutorial.* Programming Systems Lab, DFKI, Germany, www.mozart.org, 1998.

[Smith, 1992] Smith DR. *Transformational approach to scheduling.* KES.U.92.2, Kestrel Institute, CA, 1992.

[Solotorevsky et al, 1994] Solotorevsky G, Gudes E and Meisels A. *RAPS: A rulebased language for specifying resource allocation and time tabling problems.* IEEE Transactions on knowledge and data engineering, vol 6(5), pp 681-694, 1994.

[Sosic and Gu, 1994] Sosic Rok and Gu Jun. *Efficient local search with conflict minimization: a case study of the n-queens problem.* IEEE Transactions on knowledge and data engineering. Vol 6, No 5, October 1994.

[Sosic and Gu, 1991] Sosic Rok and Gu Jun. *3,000,000 queens in less than one minute.* SIGART Bulletin, vol 2(2), 1991.

[Spears, 1993] Spears WM. *Crossover or Mutation?.* In [Whitley, 1993], pp 221-238, 1993.

[Sterling et al, 1999] Sterling T, Salmon J, Becker DJ and Savarese DF. *How to build a Beowulf: a guide to the implementation and application of PC clusters.* MIT Press, 1999.

[Sterling, 2001] Sterling T. *Beowulf PC Cluster Computing with Windows and Beowulf PC Cluster Computing with Linux.* MIT Press, Cambridge, MA, 2001

[Thangiah, 1991] Thangiah SR. *GIDEON: A genetic algorithm system for vehicle routing with time windows.* Phd Thesis. North Dakota State University of Agriculture and Applied Sciences. 1991.

[Wagner, 1982] Wagner HM. *Principles of Operations research – with applications to managerial decisions* (second edn). Prentice Hall of India. 1982.

[Whitley, 1993] Whitley LD (ed). *Foundations of genetic algorithms – 2*, Morgan Kaufmann, 1993.

[Zweben and Fox, 1994] Zweben M and Fox MS. *Intelligent scheduling*. Morgan Kaufmann, 1994.

[Zweben et al, 1994] Zweben M, Daun B, Davis E and Deale M. *Scheduling and rescheduling with iterative repair*. In [Zweben and Fox, 1994], pp 241-255.

Index