University of Strathclyde

Department of Electronic and Electrical Engineering

Communications Division

Genetic algorithms for large resource allocation problems.

Claude Muller

April 1995

Submitted to the University of Strathclyde
for the Degree of Doctor of Philosophy.

Abstract

Telecommunications management systems must undergo some radical transformations in order to support the next generation of giant gigabit networks. In particular, Artificial Intelligence (AI) will be necessary to automate some aspects of network management. This thesis concentrates on one particular aspect of management, namely resource management.

A large number of telecommunications applications can be viewed as resource allocation problems and the telecommunications operators require efficient techniques to solve these problems. However, due to the complex nature of these scheduling problems, large instances cannot be tackled easily by conventional techniques such as the deterministic and analytical procedures provided by Operational Research. Alternative approaches such as the ones proposed by AI have to be considered.

This thesis examines genetic algorithms (GAs) - one of these AI techniques - applied to vehicle routing problems with time-windows and technological constraints - a special class of resource allocation problem. In particular, this thesis proposes a new class of GAs. By working directly on the schedules rather than using an indirect representation, these GAs overcome some important limitations shown by traditional GA-based systems.

These direct GAs are examined over a wide range of problems. In a first set of experiments, the direct GAs totally dominate the traditional GA-based systems. In a second study, which compares the performances of these new GAs with the performances of random search, hill-climbing, simulated annealing and tabu search, it is shown that there is no universally best algorithm. However, under certain conditions some algorithms should be preferred to others. In particular, when time is available, the direct GAs should be chosen for the solution of under-resourced problems.

Acknowledgements

I wish to thank Professor D.G.Smith for his support and encouragements during the past four years. Also, I would like to thank Dr Evan Magill and Dr Patrick Prosser whose advice, knowledge and critical supervision have been invaluable. Special thanks also go to my two partners in this project, Chris and Craig. Thank you for your help and friendship. Thanks, also, to Professor J.R. McDonald, and to Professor P. Mars of Durham University, for acting as internal and external examiners respectively.

I would also like to thank David, Lee and Grace - Glasgow would have been a different experience without your early friendship. Just a shame that I have to support United now!!! Also, I have been lucky enough to be part of a very friendly group, thank you guys. I am so sorry, Peter, that 5-a-side is not our strong point. May be, we should practice a little bit more, and go to Caffé Qui a little bit less. Thanks are also due to Alasdair, Sheona, Iain, Neil, Lois, and Lorna's Mum and Dad - 35 Kessington Road has been a great place to relax and be comforted.

Je voudrais également remercier mes parents pour leur nombreux encouragements et leur assistance tout au long de ces 28 années. Merci à vous pour vos efforts. Finally, I would like to thank my wife, Lorna, for all her help, support and patience. What an understatement !!! Thank you very much, Lorna. Promise, I will try not to be so impatient in the future ...

Contents

List of Figures

List of Tables

Acronyms

AI	Artificial Intelligence
ATM	Asynchronous Transfer Mode
AT&T	American Telephone and Telegraph Corporation
BISDN	Broadband Integrated Services Digital Network
CAP	Channel Assignment Problem
CCITT	Comite Consultatif International Telegraphique et Telephonique
CIM	Computer Integrated Manufacturing
CNA-M	Cooperative Networking Architecture-Management
CSP	Constraint Satisfaction Problem
DAI	Distributed Artificial Intelligence
DAR	Dynamic Anticipatory Routing
DCF	Data Communications Function
EDS	Electronic Data Systems
FIFO	First In - First Out
GA	Genetic Algorithm
GUI	Graphical User Interface
HC	Hill-climbing
IN	Intelligent Network
ISO	International Standards Organisation
ITU	International Telecommunications Union
ITU-TSS	International Telecommunications Union - Telecommunications Standards Sector
MF	Mediation Function
NEF	Network Element Function
NT&T	Nippon Telephone and Telegraph Corporation

OSF	Operations System Function
PMX	Partially Mapped Crossover operator
PTT	(Ministry of) Posts, Telegraphy and Telephony (a general term for a public telecommunications operator)
QAP	Quadratic Assignment Problem
QoS	Quality of Services
RACE	Research and development in Advanced Communications for Europe
RBOCs	Regional Bell Operating Companies
RS	Random Search
SA	Simulated Annealing
SDH	Synchronous Digital Hierarchy
SIP	Societa Italiana per l'Esercizio delle Telecomunicazioni
TINA-C	Telecommunications Information Network Architecture Consortium
TMN	Telecommunications Management Network
TS	Tabu Search
TSP	Travelling Salesman Problem
VC	Virtual Channel
VLSI	Very Large Scale Integration
VP	Virtual Path
VRP	Vehicle Routing Problem
WSF	Work Station Function

Chapter 1

Introduction.

1.1 Objective.

This thesis investigates Genetic Algorithms (GAs) for solving large-scale resource allocation problems, with special interest for problems in the field of telecommunications.

A large number of telecommunications applications can be viewed as resource allocation problems; for instance:

- Work force management - this is concerned with the optimal allocation of a work force to a set of construction or maintenance operations.
- Frequency allocation - the bandwidth employed for mobile phone is a limited resource and, therefore the frequencies must be allocated efficiently in order to maximise the number of channels.
- Allocation of bandwidth for video broadcasting networks.
- Design and dimensioning - given a description of network requirements and

a description of the available resources, it is possible to ensure that proposed designs meet a specified level of service.

An automated and efficient management (or allocation) of both the workforce and of the elements composing the network offers the opportunity to:

- reduce the cost of operating the network,
- improve the quality of service,
- provide service to a larger number of customers,
- compete in an open environment,
- and finally, increase profits.

Therefore, an efficient solution of this class of problems, both in terms of quality and running time, is of major concern to telecommunications operators.

In this field of research, a major difficulty is that large problems are intractable. Although conventional techniques, such as the analytical procedures proposed by operational research, can now tackle problems of respectable size, there is still an important need for approximate techniques.

Basically, these conventional techniques might require exponential time in the worst case, and hence, might fail to provide a solution - for large instances - in reasonable time.

Table 1.1, taken from [GARE79], illustrates this point and this table shows how long it would take a computer to explore m^n states for different small values of m and n. It is assumed that the computer performs an instruction in a micro-second.

m	n=10	n=20	n=30	n=40	n=50	n=60
2	0.001 second	1.0 second	17.9 minutes	12.7 days	35.7 years	36.6 centuries
3	0.059 second	58 minutes	6.5 years	3855 centuries	$2 * 10^7$ millennia	$1.3 * 10^{12}$ millennia

Table 1.1: An example of exponential complexity.

By contrast, some instances of the problem used for the experiments (see Chapters 4-5-6) have an estimated search space of more than 10^{500} states[1].

Clearly, an exhaustive exploration of such large search spaces cannot be performed in reasonable time. Systematic or deterministic algorithms cannot, realistically, guarantee that they will find the optimal solution of large instances. Observe that, even a significant acceleration (2 or more orders of magnitude) of computer speed would not be sufficient. These problems are simply intractable by nature.

Therefore, alternative approaches such as the ones proposed by Artificial Intelligence (AI) have to be considered - approaches which sacrifice optimality for efficiency.

1.2 The case for Genetic Algorithms.

A Genetic Algorithm [2] is an AI search technique. More precisely, it is a stochastic procedure, i.e. random-based decisions take an important part in the search process. This particular property has been one of the principal motivations for using GAs to solve resource allocation problems.

Indeed, their stochastic nature might permit GAs to be more efficient than deter-

[1] By comparison, the universe has an estimated size of 10^{80} atoms.

[2] Section 3.2.2 provides a brief tutorial on genetic algorithms.

ministic algorithms for handling the complex nature of some resource allocation problems (see Section 3.1 for further explanation) and generally, GAs are regarded as being an efficient means for the rapid exploration of large and complex search spaces [GLOV87] [DAVI91a].

The application of GAs to resource allocation per se is not novel. However, this research proposes a GA which overcomes some important limitations of many existing GA-based systems. In such systems, the GA works with an intermediate representation and is used in association with a solution-builder (see Section 3.2.5). The GA manipulates order-based chromosomes and the solution-builder uses these order-based chromosomes to create schedules. These systems have a number of weaknesses:

1. The indirect representation might not allow total exploration of the search space, i.e. the intermediate representation might mask some parts of the real search space.

2. This indirect approach has been applied successfully to a wide range of sequencing problems. However, this technique is likely to be disruptive when applied to problems which are not pure sequencing problems.

3. This indirect approach means that, for each new chromosome, a new solution is entirely created from scratch and this new solution is also entirely evaluated from scratch. These two operations are extremely expensive in terms of computational effort.

This research proposes and investigates a class of GAs which work directly on the candidate-solutions, i.e. the solutions are the chromosomes. During the reproduction process, part of two solutions are mixed together to produce a new solution. Therefore, there is no distortion due to an intermediate representation and the search process is also accelerated as the GA is no longer generating and evaluating

a new solution entirely from scratch for each new chromosome.

The performances of these new GAs are studied over a wide range of vehicle routing problems[3] (VRPs) - a special class of resource allocation problem, and compared against the performances of other search techniques, namely random search (RS), hill-climbing (HC), simulated annealing (SA), tabu search (TS) and also some GAs using traditional (i.e. indirect) crossovers. VRPs have been used for these experiments because the sponsor of this research project, British Telecom, is particularly interested in finding efficient tools to solve these problems.

In this particular study, direct GAs totally dominate the indirect candidate over the entire range of problems. Furthermore, the study shows that there is no universally best algorithm in the set {GA,HC,RS,SA,TS}, but it is shown that under certain conditions, some algorithms should be preferred to others.

When time is available, GA should be preferred for the solution of under-resourced VRPs. However, when the algorithm is only given a limited amount of time or when the VRP is over-resourced, then SA should be preferred.

1.3 Thesis outline.

This thesis contains 7 chapters (including this introduction). Chapters 2 and 3 both present some background material that readers familiar with telecommunications and/or AI may wish to skip. Chapters 4 to 6 concentrate on the different studies and Chapter 7 concludes this thesis. The remainder of this section now gives a brief overview of each chapter:

- Chapter 2, *"AI & Telecommunications"*, examines the relationship between AI and telecommunications and highlights the necessity for the use of AI in

[3]These problems are introduced in Chapter 3.

network management systems.

- Chapter 3, *"Novel Approaches to Resource Allocation"*, focuses on three search techniques: genetic algorithms, simulated annealing and tabu search and reviews some of their applications to large resource allocation problems.

- Chapter 4, *"Preliminary Studies"*, presents two early projects and draws some initial conclusions. In the first project, an indirect GA is applied to a job-shop scheduling problem and in the second study, an indirect GA is used to solve a particular instance of the vehicle routing problem.

- Chapter 5, *"Experimental Set-up"*, gives some detailed information (a) on the 36 problems used for the experiments, (b) on the model used to encode the solutions, and (c) on the cost function. Finally, this chapter also introduces and explains a new class of direct genetic algorithms especially designed for vehicle routing problems.

- Chapter 6, *"Experiments and Discussions"*, examines the performance of the direct GAs introduced in the previous chapter. This chapter reports three sets of experiments.

- Chapter 7, *"Conclusion"*, summarises the work that has been presented in the previous chapters, discusses areas of future work, and concludes this thesis.

Chapter 2

AI & Telecommunications.

This chapter examines the relationship between two rapidly evolving technologies, Artificial Intelligence and Telecommunications, and highlights the necessity for AI in the next generation of management systems.

The Information Revolution has already started and the pace of this revolution will accelerate in the next decade. The key component of this change, the telecommunications network, will have to undergo some rapid and profound transformations. Novel approaches such as AI will be necessary to automate some aspects of network management.

2.1 The Information Society.

This section highlights the importance of information for the social and economic developments of our society, and describes the rapid changes faced by the telecommunications industry as a result of this information revolution.

The information revolution started 150 years ago with the invention of telegraph

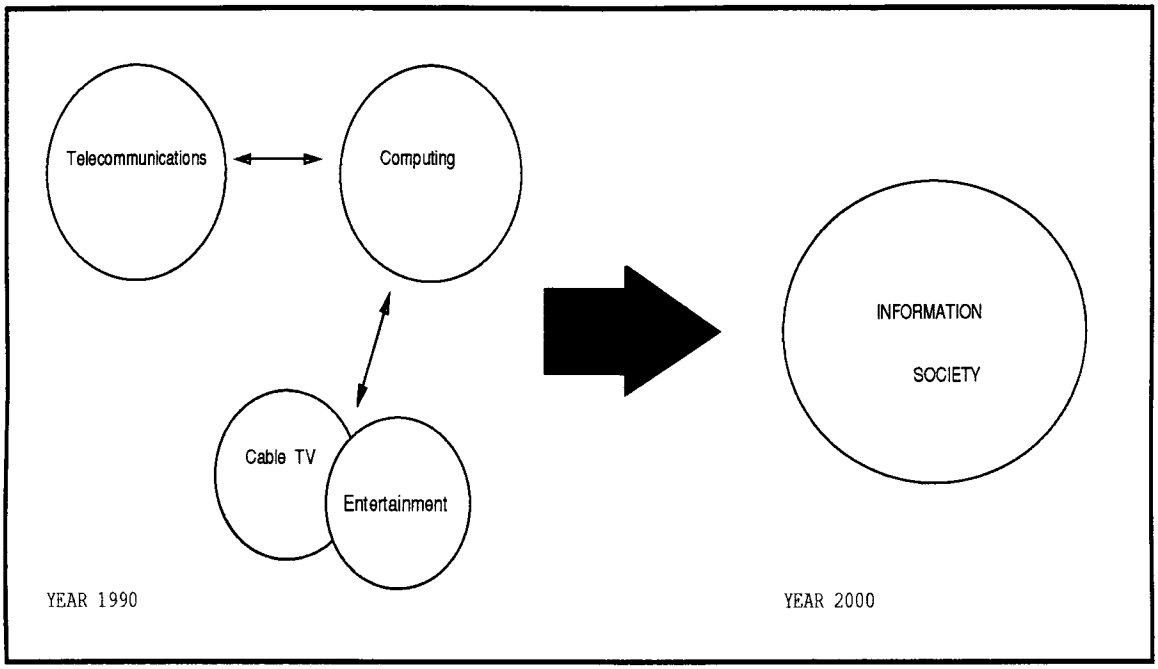

Figure 2.1: The Information Society.

and has known three main periods: the Wire Age (1844-1900), the Wireless Age (1900-1970) and we are now entering the Age of the Information Society [DIZA89].

As illustrated in Figure 2.1, this information society can be described as the coming together of the telecommunications, computing, cable television and entertainment industries [RUDG93]. More precisely, the principal characteristic of this new Age is the convergence (or synergy) between telecommunications and a number of computing-related industries.

The Intelligent Network (IN) with its intensive use of databases and its flexible architecture is a fine example for illustrating this convergence. In fact, the entire network is becoming more and more like an enormous software package [MAYO90]. Even at the lowest level, i.e. in the digital switch itself, the intelligence has moved from the hardware platform to the software. Indeed, some telephones exchanges are now using over 12 million lines of code [JOHN89a].

In order to accelerate the pace of this convergence, different industries (e.g. telecom-

munications, computing and cable TV companies) have come closer together in the recent past. Indeed, there have been many cross alliances, for instance, between (1) telecommunications and software companies and (2) telecommunications and cable TV companies [1]:

- France Telecom and Cap Gemini Sogeti (Europe's biggest software house)
- SIP (Italy's operator) and Finsiel (Europe's second biggest software house) [ECON93a]
- McCaw Cellular Communications (America's largest cell-phone operator and a subsidiary of AT&T) and Microsoft
- Sprint (America's third-biggest long-distance telephone operator) and Electronic Data Systems (EDS, a subsidiary of General Motors and the world's biggest computer services group) [ECON94].

Another example of this convergence is the consortium, named TINA-C (Telecommunications Information Network Architecture-Consortium), recently launched by BT[2], Bellcore and NT&T, which aims at regrouping equipment suppliers, computer makers, and software houses, and transforming telecommunications networks into information networks [COMM92a].

The origins of this convergence lies in the fact that the importance of information (and its processing) in our society is increasing rapidly; information is becoming a strategic resource as vital as energy. It is likely that, in the near future, a large section of the workforce will either create, process or disseminate information [RACE90]. This means that the weight of the information-based industries - and this is especially true for the telecommunications industry - within the economy is

[1]The merger of Bell Atlantic with Tele-Communications International (TCI) has been widely publicised, but this merger collapsed in April 1994.

[2]formerly British Telecom.

increasing rapidly. Already, in 1993, the world's two most valuable companies were NT&T and AT&T with a respective market capitalisation of $146 billion and $85 billion [DUBR93].

In Europe alone, broadband services could generate annual revenue for the carriers of about $13.1 billion by the year 2000 [RACE90]. By that time, the European Union expects the telecommunications industry to account for more than 7% of its Gross National Product [ECON92a].

In order to meet these expectations, the telecommunications industry must undergo some radical transformations. Indeed, while the computer industry has enjoyed an extremely rapid growth, the pace for the telecommunications industry so far has been slower [RACE90] and the lack of efficient means of telecommunications still represents an obstacle in the development of the information society.

However, this situation is changing rapidly for political and economical reasons. North America has lost its leadership in consumer electronics while Europe has lost its position both in consumer electronics and in the computer industry. Therefore, these two players cannot afford to lose their leading positions in the telecommunications industry.

Europe traditionally has had an important share of the telecommunications market with successful companies such as Alcatel, Siemens, Ericsson, GPT and Philips. In 1991, these European companies sold nearly 55% of all the telecommunications equipment worldwide with the remaining 45% shared between Japan (15%) and North America (30%) [BELL93].

Europe must maintain this strong position despite the intensification of the competition. As a consequence, the European Union launched in 1987 a massive research initiative, RACE (Research and Development in Advanced Communications for Europe) for the purpose of developing technological solutions (e.g. optical switching systems, digital mobile systems) which aimed at introducing cost-effective broad-

band services by the year 1996 [RACE90].

There is a second force which motivates the development of a successful telecommunications industry: corporate users have indeed discovered how to use information technologies as strategic tools. In other words, the ability to process and transfer information quickly and accurately is becoming essential in order to maintain the competitiveness of a business [WILL91]. For instance, the credit-card group VISA implemented recently, a telecommunications system which demonstrates just how important telecommunications services can be as corporate tools. This system, called *PS 2000* , is described as *"a vast information super-highway"* with two computer centres and 9 million miles of optical fibre connecting 18 400 banks around the world. The total cost of this system was around $40 million; however, it is expected that the *PS 2000* system will reduce the cost resulting of counterfeiting from $125 million to $20 million in just one year. Overall, the banks were expected to save some $270 million in 1993 [ECON93c].

This example shows that private networks, and the associated services and information management facilities that they provide, are becoming increasingly important in the management of a company. Hence, from a company's point of view, the effective management of telecommunications services represents a strategic issue affecting profitability [WILL91].

Corporate users want the telecommunications industry to answer their needs, i.e. they want a telecommunications market driven by the users. Many believe that this will only be possible through the creation of a competitive market. Hence, many large companies are putting pressure on governments and PTTs to increase competition[3]. Until recently, a real competitive market was not a possible option. In the USA, AT&T was broken up in 1984 and it still had 66% of the long distance

[3]For instance, Fiat, Europe's second biggest car maker, is lobbying the European Union; it claims that it could cut its $100 million-a year phone bill by at least $43 million if the European telecommunications market was liberalised.

traffic in 1992. In the same year, the regional Bell operating companies (RBOCs) controlled, on average 75% of the local traffic [COMM92b]. In the United Kingdom, the duopoly between BT and Mercury was also started in 1984 and BT still had a 92% share of the market in 1992 [ECON92b]. Basically, the investment required for the realisation of a physical network is immense and represents a considerable barrier for any new competitor.

However, the deployment of new technologies such as cable TV, satellites and cellular services allow new entrants to overcome this obstacle. For instance, in [ECON92b], the author claims that a high capacity digital wireless network covering the whole of Britain could cost under $2 billion to build (around one order of magnitude less than the original copper network).

To illustrate this new situation, AT&T has recently made a strategic alliance with McCaw Cellular Communications which has a cellular network covering more than 40% of the US population. Such an alliance permits AT&T to re-enter the local loop market and compete with the RBOCs [COMM92b] [ECON92c]. AT&T, MCI and Bell Atlantic have contracts with cable television companies that will allow them to attack very lucrative sectors of telecommunications; video and data communications [BELL93] [ECON93b]. In the United Kingdom, a number of companies will rapidly challenge the duopolists (i.e. BT and Mercury); for instance, Energis (a telecommunications company owned by the National Grid company), Ionica, British Rail Telecommunications, and British Waterways. On top of this, 39 cable television companies have been granted a licence by the British government to offer telecommunications services [FAGA93].

Thus, the telecommunications market will, very soon, be driven by the users' requirements and the principal demands will be service quality, open access, flexibility, mobility and lower cost. More precisely, Nicholas and Yeomans, in [NICH93], give the following list of basic requirements:

- availability: same systems at any point of requirement, same operator at any point of requirement, single point of accountability.
- open access: ability to inter-operate between systems, ability to inter-operate between providers, ability to choose between systems or providers.
- open use: no utilisation restrictions, application independent, service independent, technology independent.
- rational prices: transparent, non discriminating, commercially realistic.
- accountability: contractual, commercial, defined conditions, dispute process.

These demands introduce major challenges to all strands of the telecommunications industry [FITC91] and in order to meet these demands, the operators must transform their networks radically. The three branches of telecommunications, i.e. transmission, switching, and management, will have to undergo significant transformation. In the near future, SDH [4](for transmission) and ATM [5](for switching) will provide a network infrastructure capable of offering a wide range of services with diverse traffic characteristics (e.g. bandwidth requirements, delay and loss sensitivity, and burstiness level). In order to support this infrastructure, the way the operators manage their networks will need to be revised entirely [HELL91].

2.2 Network Management.

To quote from [GUID90], *"network management is about the people, procedures and tools required effectively to manage a telecommunications network at each stage in its life cycle (i.e. pre-service, in-service and future service)."* Network management can also be defined as the tasks of:

[4]SDH stands for Synchronous Digital Hierarchy.
[5]ATM stands for Asynchronous Transfer Mode.

1. maintaining an efficient and reliable network,
2. increasing performance both in terms of quality and quantity,
3. while using a minimum of network resources.

Network management may, in fact, be viewed from different perspectives. For the owner, the network is a business, and the primary objective is to provide as much service whilst using the smallest possible investment (i.e. maximise profits); for the operator, the network has to be robust, easy to maintain, analyse and expand and finally, for the user the network must provide (as seen before) service quality, open access, flexibility, mobility and low cost [LIEB88].

Network management must accommodate these different demands.

2.2.1 Functional Division.

The problem with network management is that it is an enormous and complex task which cannot be tackled by a single agent; there has to be a division of the tasks and ISO (International Organisation for Standardisation) has separated the different management functions into five classes [KLER88]:

- **Accounting and commercial management** determines and allocates costs, and charges for the use of the network elements.
- **Performance management** monitors, evaluates and optimises the performance of the network.
- **Configuration management** maintains precise information about the identity and status of every network element.
- **Security management** provides access protection and control over the network elements.

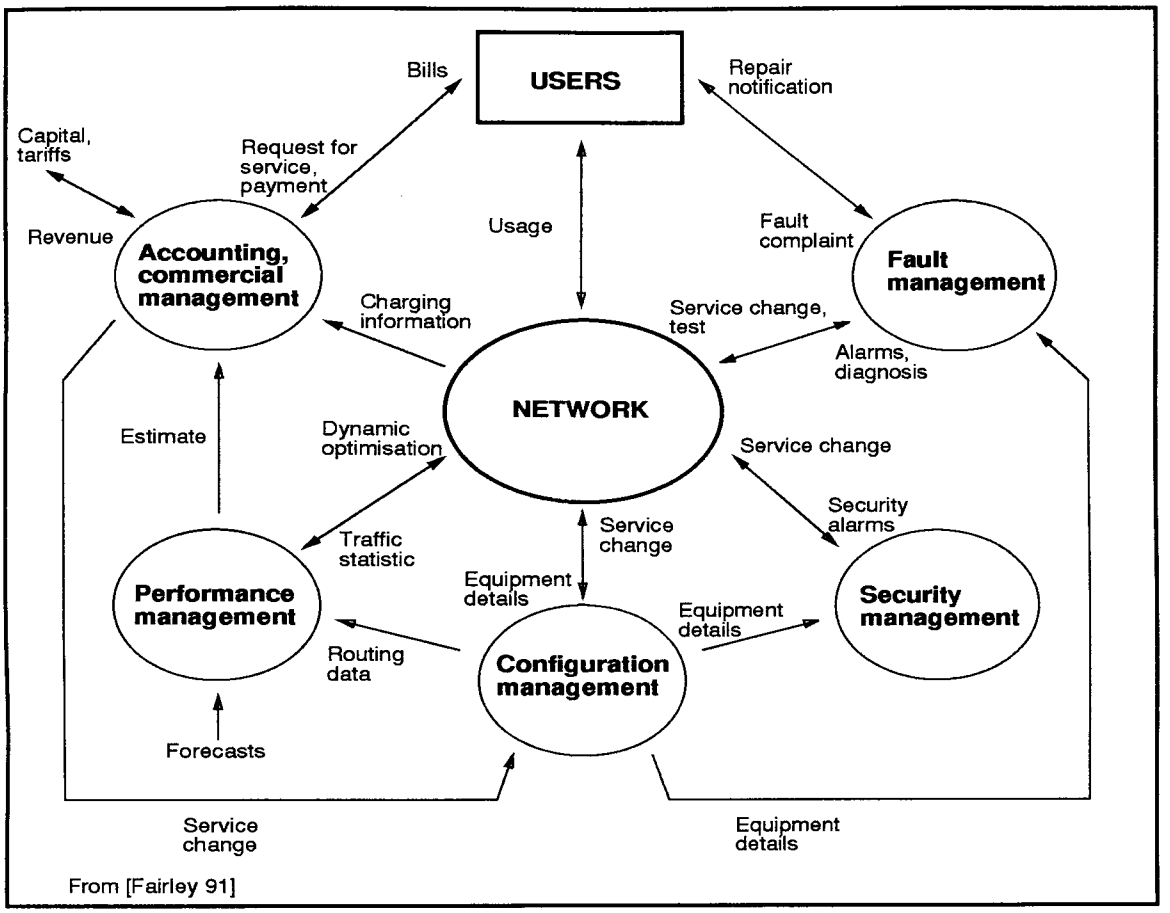

Figure 2.2: Network management.

- **Fault management** is concerned with detection, diagnosis, isolation and repair of faults occurring in the network.

Figure 2.2, from [FAIR91], provides a global view of network management; this diagram also shows the relationships and the flow of data between these different functions.

2.2.2 Telecommunications Management Networks.

The management functions discussed in the previous section are performed by individual operations systems. In order to achieve a global, coherent and integrated management, there is also a need for a management network which would provide the ability for the different operations systems to exchange information and interact.

The CCITT[6] (Comité International Télégraphique et Téléphonique) Recommendation M.30 proposes a telecommunication management network (TMN) to facilitate the exchange of information between operations systems and network elements [CCIT88].

TMN will enable an operator to manage its own network efficiently but it will also enable different operators to exchange information, to support interaction and to encourage mutual co-operation. This network will have a crucial part to play in the successful deployment of a competitive market, where several companies will operate, where network control will take place in an open co-operative environment (across geographical and technological boundaries) and finally, where the different actors will have to interact and exchange information.

M.30 proposes five functional blocks:

- **Operations System Function (OSF)**: performs the different management functions.
- **Mediation Function (MF)**: provides the links required between the network elements and the operations systems.
- **Data Communications Function (DCF)**: transports management mes-

[6]Following a reorganisation of the ITU (an agency of the United Nations dealing with telecommunications) in 1993, the CCITT has now been replaced by the ITU-TSS (international Telecommunications Union - Telecom Standards Sector).

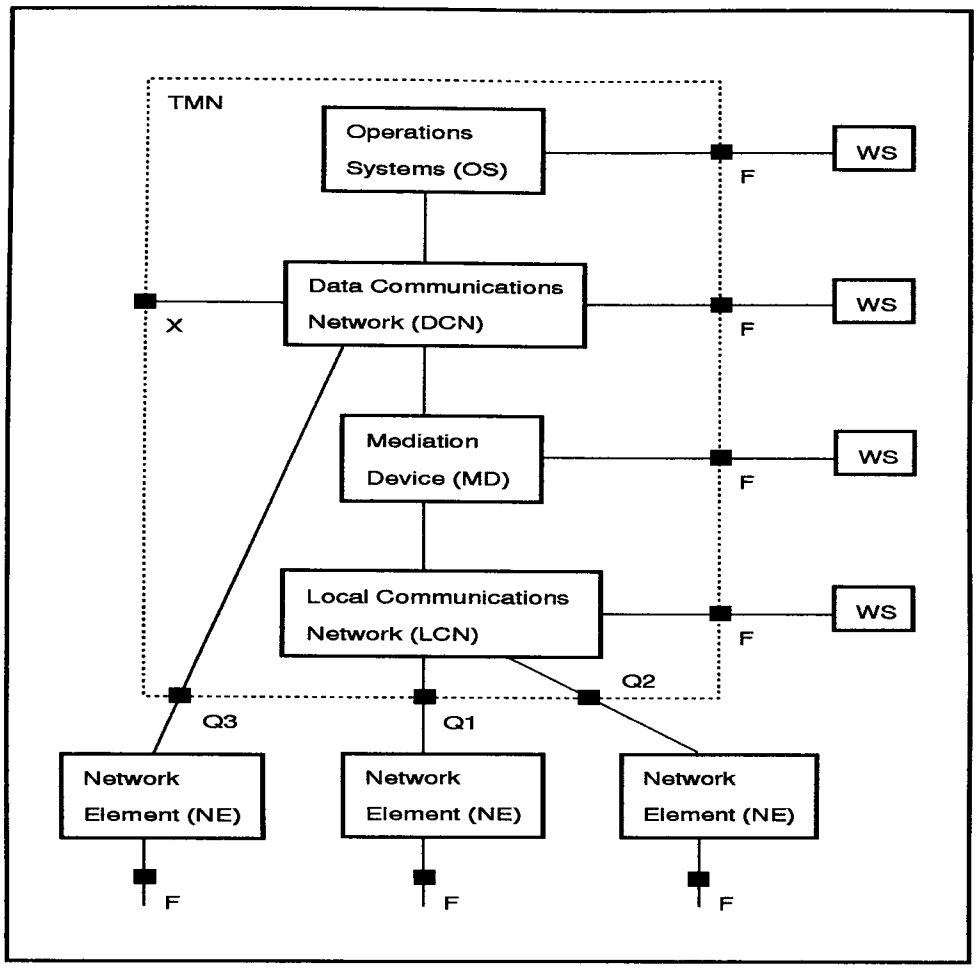

Figure 2.3: TMN.

sages between the different elements of the TMN.

- **Network Element Function (NEF)**: the different elements which make up the network.
- **Work Station Function (WSF)**: provides an interface between the operator and the TMN.

The interconnections between the various TMN elements (i.e. NEF, MF, and OSF) are made possible by the Q.1, Q.2, and Q.3 interfaces, either directly or via the DCF. The F interfaces link the OSFs either to the WSFs or to the MFs. The G interfaces (not displayed on Figure 2.3) provide the interfaces between the operator

and the TMN, more precisely between the users and the WSFs. Finally, the X interfaces support the dialogue between several separate TMNs [GUID90].

In addition to TMN, the ISO/NM Forum also proposes a framework for the co-operative management of a full range of networks [EMBR90]. Both architectures provide a communications infrastructure to allow for the transmission of monitoring and control data.

2.2.3 Network and Service management.

A layered view of network management is also required: operators can no longer simply maintain their networks and must also manage efficiently the numerous services provided by both themselves and other parties. BT's CNA-M architecture [WILL91] and RACE [SMIT91] have introduced the concept of service management, which is expressed as a distinct layer above network management. A telecommunications service may involve several networks (e.g. voice and data) and several operators (e.g. due to national boundaries). The service management layer will integrate the traffic, manage the underlying networks and perform the interface between customers and operators.

The CNA-M (Cooperative Networking Architecture-Management) is shown in Figure 2.4. Essentially, CNA-M divides network management into a hierarchy of management layers: business, service, network, and element management. The peak of the pyramid indicates minimum operator involvement, while the base of the pyramid indicates maximum operator involvement.

Each layer performs a well-defined class of functions. As seen earlier, the "service layer" is responsible for the interface between the customers and the operator. In [SINN91], this layer is defined as "the layer which must provide, create, monitor, inform, maintain and account for the services of each customer". The service layer

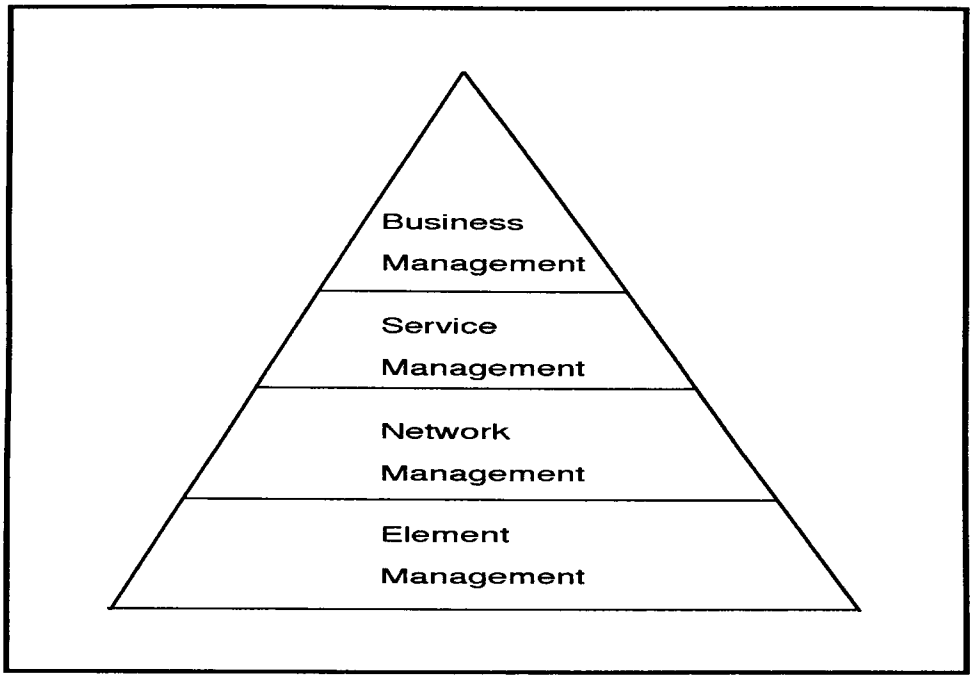

Figure 2.4: A layered view of network management.

requests its information from the network layer. It also provides data to the business layer, which monitors and manages the overall business.

This section has presented several propositions which should permit operators to manage their networks in an efficient manner. However, the increasing complexity of the network and its associated services means that a reduction of cost and the improvement of quality can only be achieved through a new level of automation [MAMD91]. Modern systems can be monitored comprehensively, but the resulting large volume of data must be filtered to identify relevant information. Conventional computer applications provide a degree of automation to this process but, since much of the data is incomplete and conflicting, human interaction remains essential.

In principle, Artificial Intelligence could limit the need for human intervention, and so increase the level of automation [HASK90]. Hence, the application of AI to management has appealed to many operators. The next section will focus on this issue, and discuss the use of AI to help automate aspects of management (at each

layer in the management hierarchy).

2.3 AI in Network Management.

> "Artificial Intelligence is the science (and art) of developing computer programs that emulate human-like thinking and reasoning. Unlike conventional software that primarily manipulates data, AI programs depend critically on the manipulation of knowledge."[CEBU89]

To quote from [BODE90], *"Artificial intelligence was conceived in the mid-1940s, born in the early 1950s, and christened in 1956 (at a small meeting of computer scientists, psychologist, and physiologists)"*. Since then, AI has generated an important debate and a large section of the scientific community does not perceive AI as "real science", they would rather speak of the myths of AI (see [WEIZ90], [PENR89]).

A possible reason for this, is that the expectations have always been far too unrealistic. For instance, Herbert Simon and Allen Newell wrote in 1958:

> "There are now in the world machines that think, that learn, and that create. Moreover, their ability to do these things is going to increase rapidly until - in the visible future - the range of problems they can handle will be coextensive with the range to which the human mind has been applied." [SIMO58]

In the middle of the 90's, this assertion is still unrealistic; indeed, not many 1994-AI systems would satisfy this statement. In fact, at one time, the critics were very important; so much so, that, in 1972, Sir James Lighthill nearly killed off British research by writing in his review for the British government:

> "In no part of the field have the discoveries made so far produced the major impact that was then promised (in the 1950s)." [INDE93]

Despite these problems, AI is not just a technology for the future; in fact, it can offer very real benefits today and some products using AI technologies, are now emerging. In 1993, the AI industry represented a market of over $600 million in America and has reached a mature position [HARM94]. Various AI techniques are now considered for the new generation of management systems. For instance, RACE had 6 projects investigating the applicability of advanced information processing techniques (including AI techniques) to network management in the areas of maintenance, network and customer administration systems, and traffic and quality of services (QoS) management [SMIT91].

AI techniques offer a number of advantages over traditional methods; in particular, they can address ill-defined problems (e.g. missing and/or incoherent data), solve problems that conventional algorithms find intractable (e.g. due to combinatorial explosion) and, in some systems, can learn from past experience.

AI encompasses a wide range of techniques (e.g. knowledge acquisition and representation, case-based reasoning, machine learning, robotics, simulation, virtual reality, etc.). Figure 2.5 illustrates a limited subset of these techniques. In this figure, AI has been divided into a number of classes. The first class is concerned with high-level tasks such as planning, design or diagnosis; it includes expert systems, model based reasoning, constraint satisfaction, and also, natural language and human-computer interfaces[7]. The second class, neural networks, is less procedural and is more concerned with low level tasks such as perception and pattern recognition. The third class, Distributed Artificial Intelligence (DAI), attempts to solve a problem with a collection of agents rather than with a single agent. The last class, Meta-heuristics, includes genetic algorithms, simulated annealing and tabu search. These approaches are now considered separately.

[7]Although these last two techniques will play an important part in the successful development of the next generation of network interfaces, they will not be discussed further.

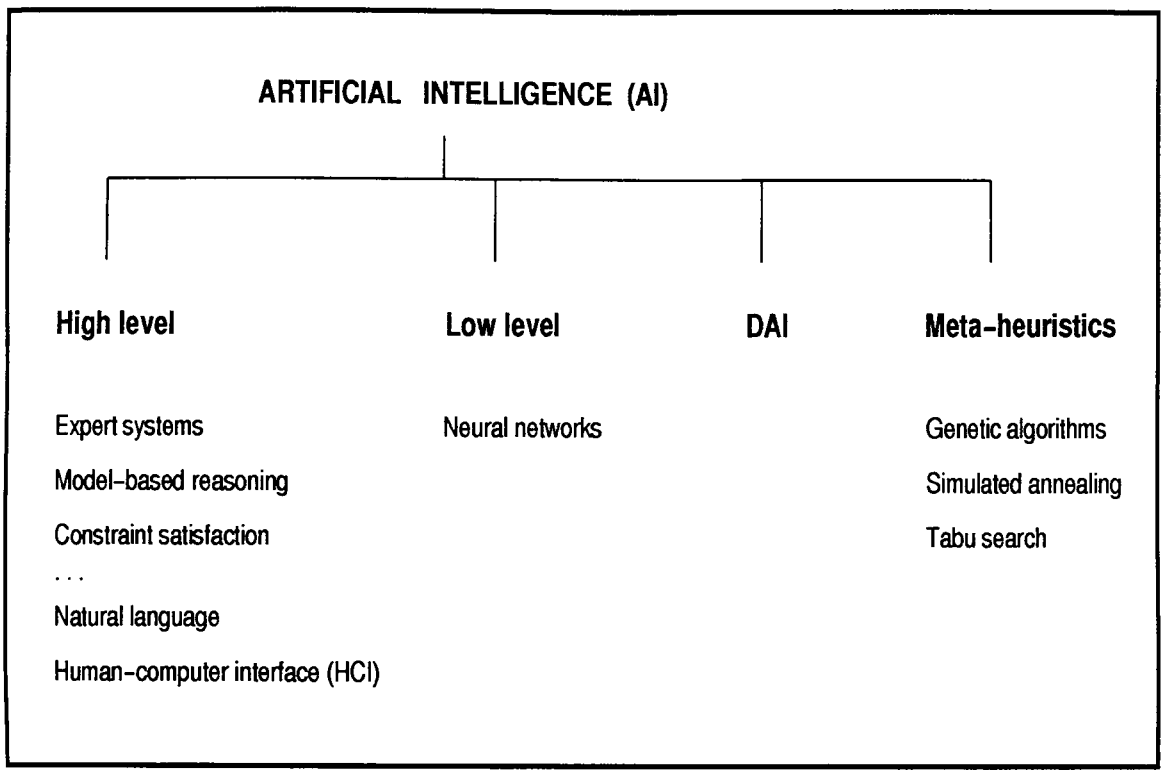

Figure 2.5: A selection of AI techniques.

2.3.1 Expert Systems.

For many companies, expert systems have represented the first step towards AI. They are employed to solve complex tasks and are only knowledgeable in a narrow domain. Under this paradigm, knowledge was originally represented as a set of "**if-then**" rules; they represent the rules of thumb, heuristics employed by the human field-experts to solve their problems.

The principal advantages of expert systems are their simplicity and the flexibility of their internal architectures which permit the rapid development of prototypes. However, the acquisition and validation of the rules represent two major problems [LIPP91]. Also, expert systems have often been developed as prototypes or stand-alone systems and they suffer from a lack of integration in their environments [8]

[8]This might also be the case for all the other AI-based systems.

[WRIG90]. Finally, their complexity means that expert systems cannot be considered for time-critical operations. All these limitations reduce the applicability of these products to real world problems.

However, ranging from diagnosis through planning to traffic management applications, a large number of expert systems have been developed. In [LIEB88] and in [MALC90], the authors provide general surveys, while design applications and maintenance applications are reviewed in [LIRO91a] and [LIRO91b] respectively.

2.3.2 Model-based Reasoning.

Model-based systems [LIPP91], [MANN90] represent the second generation of AI systems and attempt to overcome the limitations encountered by expert systems. Currently, these systems can mainly be found in maintenance and diagnostic applications [LOES90].

These systems reason with a model of the network under control. This model contains all the necessary information comprising information about the internal behaviour of the components and the dependencies between them. For example, given a set of fault symptoms reported by the network elements, the model-based system reasons with the model and generates a set of fault hypotheses. These fault hypotheses are then used to predict a behaviour (i.e. a set of symptoms). The predicted and the observed symptoms are then compared in order to refine and minimise the set of fault hypotheses.

Model-based systems are not limited by knowledge acquisition and their models can be modified easily. Their principal limitations are (1) the definition of the correct levels of abstraction for the different models [LIPP91] and (2) their computational complexity. Time-critical applications (e.g. traffic management) will require an important speed-up.

2.3.3 Constraint Satisfaction.

Constraint satisfaction views a problem as a set of variables, where each variable has a domain of values and a set of constraints that act between subsets of the variables. The problem is then to find an assignment of values to variables, from their respective domains, such that all the constraints acting between the variables are satisfied [TSAN93]. Constraint satisfaction can be used to solve problems which have a large number of constraints, for example, in machine vision, belief maintenance, temporal reasoning, graph problems, floor plan design [KUMA92] and also in design, planning [LUSH90] or scheduling applications [BURK91].

Unfortunately, the general CSP is NP-complete[9] and in the worst case, deterministic algorithms may require exponential time. Hence, for large instances, these algorithms may fail to provide an answer in reasonable time.

2.3.4 Neural Networks.

A neural network is based on a massively parallel architecture where many primitive processing elements interact with each other. Its principal task is to perform pattern recognition and it can be trained to recognise a set of patterns [MASS90]. After training, a neural network should be able to handle imperfect or incomplete data, thus providing a degree of fault tolerance.

A number of neural network models are currently under investigation for telecommunications applications. For instance, multi-layer perceptions have been applied mainly to speech recognition and vision [MYER90] [MCCU88]. Feed-backward networks (e.g. Hopfield networks) have been applied to combinatorial optimisation problems, large resource allocation problems [JOHN92] and traffic management problems [MAMD91] [MORR91]. Neural networks can be used on-line for real-

[9]This term is explained in chapter 3.

time applications and a number of VLSI (Very Large Scale Integration) and optical neural networks are now available [HOWA88] [PSAL90].

2.3.5 Distributed Artificial Intelligence.

DAI is the branch of AI that deals with the interaction of intelligent agents [GASS89] [ERCA91]. In other words, a DAI systems consists of a group of agents that may interact by cooperation, by coexistence or by competition [CHAI92]. Traditionally, each agent is in charge of a sub-part of the problem. Another alternative is to let each agent solve the entire problem, i.e. the same problem is given to n agents. In both cases, the agents communicate and exchange their findings during the search; different levels of communication, co-operation and control among the agents can be used to achieve a solution.

DAI represents a very promising technology for telecommunications. Several reasons (e.g. complexity of the tasks, competitive environment, and geographical distribution) will force the distribution of management responsibilities among a large number of intelligent agents [GRIF91]. Thus, the capacity to reason with incomplete, inconsistent, and distributed knowledge, the exchange of information and co-operation between local agents will be essential in order to achieve a coherent and global management. Some telecommunications applications are already using DAI techniques; Lebouc and Stern propose a general architecture for distributed network management [LEBO91], Lirov and Melaned solve design problems in a distributed manner [LIRO91a], and Garijo and Hoffman use this technique to solve operations and maintenance problems [GARI92].

2.4 Conclusion.

This section now attempts to associate the four management layers presented in Section 2.2.3 with the different AI techniques introduced in Section 2.3, i.e. decide which AI techniques might be used for which management layer.

At the lowest level (e.g. element management), the time-response becomes critical. The main emphasis is not on providing the optimal answer, but rather it is on providing a *good solution* quickly. The environment changes extremely rapidly, and if the time-response is too slow, the solution becomes irrelevant as the problem changes before the solution can be implemented. AI-based systems have to operate on-line, and they should preferably be hardware-based (like neural networks) rather than software-based. These systems have to process the information available (often incomplete and/or incoherent) and make their decision as quickly as possible, hence iterative methods are not suitable for this kind of problems. Here, the main objective of AI is to accelerate the control process, try to automate that process, and if possible remove any human intervention from the control loop.

At the highest layers of the hierarchy, the main problem for the operator is that he or she has to cope with an overwhelming amount of data. At this level, AI-based systems should be used to assist the user, as decision-support or consultative tools. They can operate off-line; the time-response is important but not critical. The main challenge is to present the user with the right information at the right time. At this level, it will be possible to use AI techniques for processing the raw data and present only the relevant information. For example, model-based expert systems can be used to hide the complexity of the network behind several layers of abstraction.

The next chapter focuses on one particular aspect of management where AI can also be particularly effective: Resource Management (e.g. network design, frequency assignment, work force management). Three meta-heuristics, namely genetic al-

gorithms, simulated annealing and tabu search are presented and some of their applications to resource allocation problems are reviewed.

Chapter 3

Novel Approaches to Resource Allocation.

This chapter investigates three iterative search techniques which offer some promising results for the solution of large resource allocation problems. These techniques are genetic algorithms (GAs), simulated annealing (SA) and tabu search (TS). In contrast with constructive techniques such as constraint satisfaction where the algorithm works with an incomplete solution and constructs the final solution piece by piece, these three methods work as follows: the algorithm starts with one (several for GA) candidate solution and proceeds by transforming this solution into another one, in an attempt to improve its quality. This cycle is repeated until certain termination criteria are satisfied.

This chapter starts by introducing the general resource allocation problem. The different techniques are then presented separately.

3.1 The resource allocation problem.

In a paper entitled *"Why scheduling is hard (and how to do it anyway)?"*, Parunak defines a schedule, i.e. one feasible solution of a resource allocation problem, in the following terms:

> "A schedule is a subset of a cartesian product of three sets. There is a set of tasks (**What**) that need to be done, there is a set of time-periods (**When**) during which a task might be performed and finally, there is a set of resources (**Where**) that tasks occupy as they execute." [PARU87]

In other words, Parunak defines the resource allocation problem as the task of deciding **What** happens **When** and **Where**.

	TSP	VRP	Scheduling	Telecommunications
What	visits	deliveries	jobs	calls
When	time-windows	time-windows	time-windows	time-frames
Where	N cities	locations of customers	machines	network elements

Table 3.1: Some resource allocation problems.

Several classes of resource allocation problems [1] are now considered and are mapped into the above definition (as illustrated in Table 3.1):

- A travelling salesman problem (TSP). In this classical problem, a salesman has to visit N cities and returns to the city of origin. Each city is to be visited once and only once, and the route is to be made as short as possible. In this case, "What" is the set of visits, "When" is the set of time-windows when the different cities can be visited and "Where" is the set of N cities.

[1] Applications of GA, SA and TS, in these 4 different problem-domains, are reviewed later in this chapter.

- A vehicle routing problem (VRP) consists of a workforce of engineers operating from one base, and a set of customers requesting a service (e.g. repair). One possible objective might be to perform all jobs while minimising the total distance travelled by the different engineers. Furthermore, this basic problem might be complicated by several side-constraints, e.g. the engineers might operate from several bases located in different parts of the map; the duration of a job might be dependent on the skill of the engineer allocated to this particular job. In this problem-domain, "What" represents the different services performed by the engineers, "When" is the set of time-windows and "Where" is the location of the customers.

  To illustrate this class of problem, Figure 3.1 displays one possible solution of a particular VRP (studied in more detail in Section 4.2). The squares represent 11 bases and the circles represent 250 jobs (geographically distributed). This figure also shows the tours of the different engineers and a set of conflicting jobs which have not been allocated.

- Scheduling tasks such as job shop scheduling, flow shop scheduling, wafer fabrication and flexible manufacturing systems; "What" may be a set of jobs to be done, "When" (as usual) may be a set of time-windows during which the jobs can be done and "Where" may be a set of machines that perform the jobs.

- As discussed previously, a large number of telecommunications applications can be regarded as resource allocation problems. In a telecommunications environment, "What" may be a set of calls with diverse traffic characteristics (e.g. bandwidth requirements, delay and loss sensitivity, and burstiness level), "When" may be a set of time-frames when the calls can be carried over the network, and "Where" may be a set of network elements - physical links, switches, multiplexers - required for the completion of the different calls.

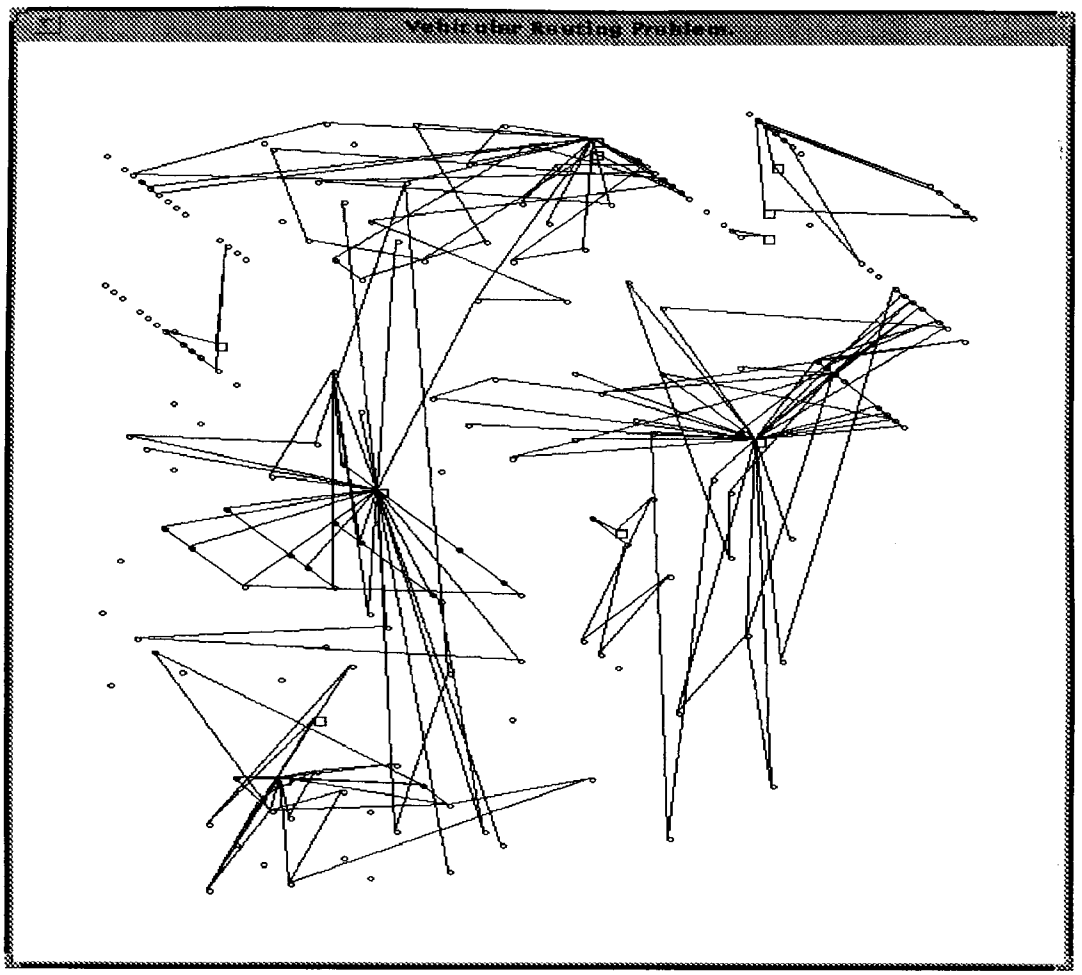

Figure 3.1: A VRP and one of its possible solutions.

Resource allocation problems have attracted a lot of attention because their efficient solution can generate massive benefits, e.g. reduction of cost and improvement of quality. However, such problems can be extremely hard to solve; typically, they are NP-hard [PARU87]. NP-hard problems are the hardest problems in the classification of Garey and Johnson [GARE79]. In this classification, combinatorial problems are divided into four classes, from P to NP-hard problems:

1. a **P problem** can be solved by a deterministic algorithm in polynomial time; given the same input, a deterministic algorithm will always follow the same sequence of instructions.

2. an **NP problem** can be solved by a non-deterministic algorithm in polynomial time; a non-deterministic algorithm can be viewed as a two-stage algorithm, first the algorithm guesses a possible solution to the problem under study and in a second stage, the algorithm verifies that guess.

3. an **NP-complete problem** is NP and is at least as hard as every other NP problem.

4. an **NP-hard problem** is NP complete, and may be harder than NP, i.e. it may require exponential time, even for a non deterministic algorithm. In this case, the algorithm may fail to provide a solution (for large instances) in reasonable time.

Observe that this decomposition is a worst case analysis for a whole class of problems but it says nothing about the complexity of a particular instance. It may be the case that particular instances are easier to solve.

Furthermore, the solution of these resource allocation problems is rendered more complicated by two other difficult tasks:

1. implementing a suitable model - or representation - of the problem.

2. designing an appropriate objective function.

The model must give a correct representation of the "real-world" problem and must allow a total exploration of the search space, i.e. the set of all possible solutions. If it provides an over-constrained [2] view of the problem, some possible solutions might not be accessible. On the other hand, if the representation is under-constrained, the search technique might have to explore an unnecessarily large search space or might create solutions which are not feasible.

[2] A problem is said to be over-constrained when too many constraints are acting on the problem and a problem is under-constrained when only a few constraints need to be satisfied.

The objective function - or evaluation function - plays a crucial part in the search process by judging and comparing the quality of the candidate solutions, and hence by guiding the search technique in its exploration of the search space.

However, resource allocation problems have many conflicting objectives and the objective function must reflect and balance the relative importance of these different objectives. For instance, in a telecommunications environment, the evaluation function may have to take into account the following objectives: minimise the worst blocking probability of a set of call classes with diverse traffic characteristics, maintain fairness among these classes, balance load on the network, maximise throughput, maintain robustness, minimise delays.

The remainder of this chapter examines three search techniques, GA, SA and TS. Each technique is described in some detail, and a number of applications are reviewed for each technique. The last section compares the three techniques and concludes this chapter.

3.2 Genetic Algorithms.

3.2.1 Introduction.

In the real world, species can adapt to changing and complex environments mainly because their evolution is dictated by two powerful mechanisms:

1. selection, i.e. survival of the fittest,
2. recombination, i.e. inheritance of genetic material.

Holland [HOLL92] used these two principles to develop GAs. In a few words, a GA is a stochastic and iterative search procedure which works with a pool of candidate

solutions. Iteration after iteration, the algorithms selects above-average solutions for reproduction and recombines them. The ultimate objective of this iterative process is to build a high quality - or optimal - solution from the information present in the pool of solutions.

Holland's original work is reported in a book entitled *"Adaptation in natural and artificial systems"* [HOLL75]. His principal claims were (1) that, given some conditions on the problem-domain, simple data structures, i.e. bit strings, could be used to represent solutions of complex problems and (2) that a repeated cycle of appropriate selections and recombinations could rapidly improve the average quality of a population of solutions.

These characteristics have advocated GAs for the solution of complex problems that conventional algorithms, e.g. hill climbing, exhaustive tree search, cannot handle easily. In particular, this technique is viewed as a promising search technique which gives the opportunity to explore large and complex search spaces in an efficient manner [GLOV87] [DAVI91a]. For instance, GAs have been applied to the following problem-domains: function optimisation [DAVI91b], travelling salesman [GOLD85] [GREF87], vehicle routing [THAN91], bin-packing [DAVI85a], scheduling [LAWT92], graph colouring [DAVI85a], [DAVI91b], routing for telecommunications networks [Cox et al 91] and multiple-fault diagnosis [LIEP91].

3.2.2 Genetic algorithms: a brief description.

How does a GA work?

A genetic algorithm works with a population of chromosomes (or strings) where each chromosome (see Figure 3.2) represents one possible solution to the problem under study; each element (or locus) of a chromosome describes one feature of the problem, and each locus can be assigned different values (or alleles):

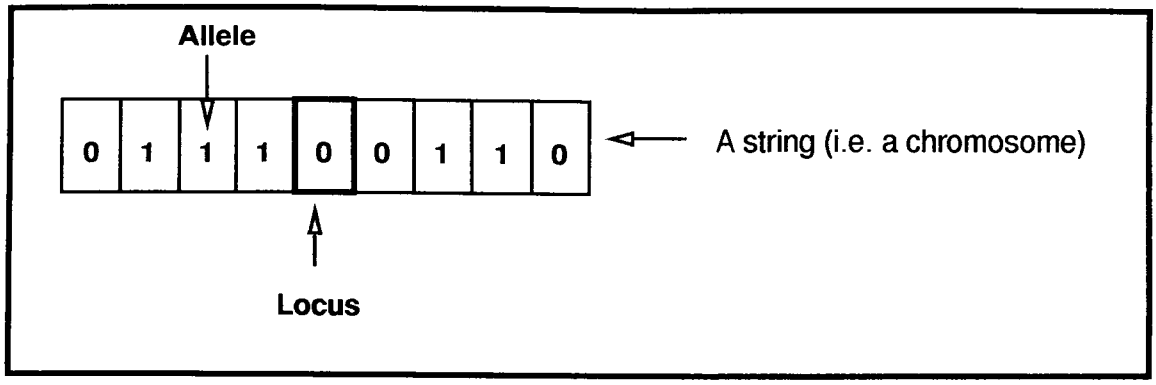

Figure 3.2: A chromosome.

Originally, the different elements could only be assigned a binary[3] value (0 or 1); the feature was either there or not. Other chromosomal representations have been designed in order to apply GAs to problem-domains where the binary representation is not suitable. Subsections 3.2.5 to 3.2.7 illustrate this last point, and in particular Section 3.2.5 discusses the applications of GAs within the resource allocation domain.

There are many different ways in which to implement a GA; however, they all share the same general principles (see Figure 3.3 for the basic GA cycle). Two possible models - the steady-state model and the generational model - are now introduced.

The steady-state model works as follows: an initial population of chromosomes is created randomly. Each chromosome is given a fitness value which is directly related to the quality of the solution that its binary string represents.

Once this initial stage is finished the algorithm enters a loop. Two chromosomes are selected with a probability which relates to their fitness; the chromosomes with the best fitness being more likely to be selected. Copies of their strings are then manipulated and recombined with different genetic-like operators, e.g. crossover, mutation and inversion, in order to create a new chromosome. Once its fitness has

[3]The binary representation was preferred because it offered the finest granularity and greatest flexibility, especially in identifying similarities between chromosomes.

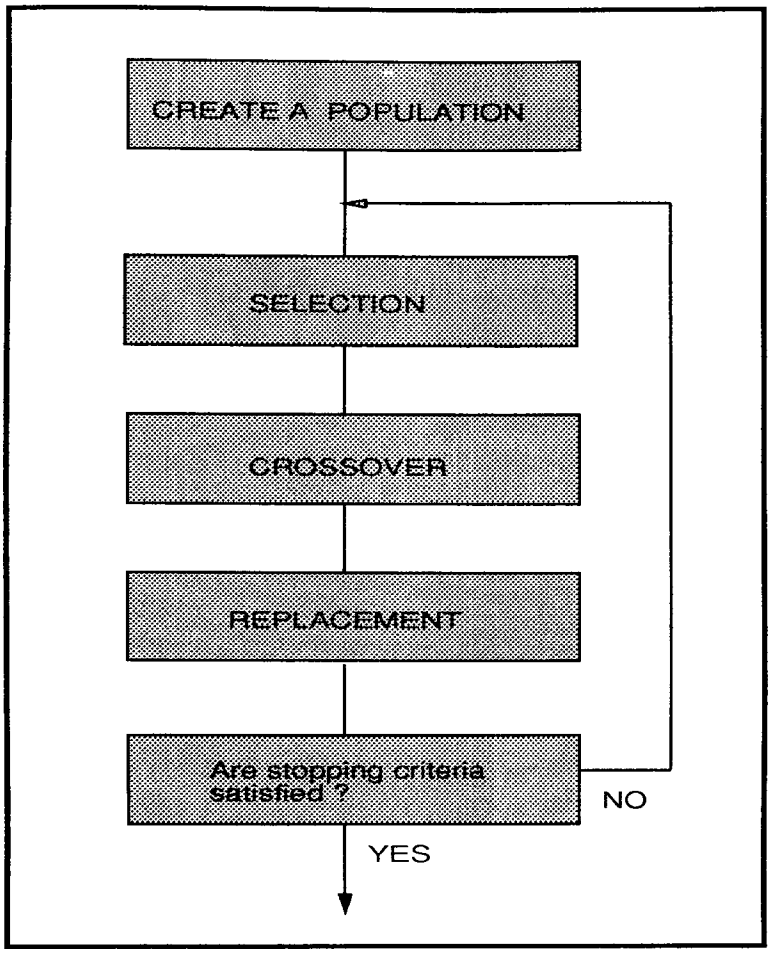

Figure 3.3: Flow of a genetic algorithm.

been evaluated, the *"child"* is inserted in the population and the worst chromosome of the population, i.e. chromosome with the worst fitness, is eliminated. This cycle "selection, reproduction, replacement" is repeated until a certain termination criterion is satisfied.

The generational model is another, more popular alternative. Under this model, the GA works with two populations: the old one and the new one. Parents (in the old population) are selected for reproduction and they generate offsprings in the new population; an entire population is generated at each population. In some cases, a certain percentage of the old population might be copied - or cloned - in the new population; the term of *generational gap* is used to describe this phenomenon.

Comparison with other search techniques.

GAs differ from other search techniques in several ways:

1. Random-based decisions play an important part in the search process; basically, the final solution delivered by a GA depends on both the initial random starting points, i.e. the initial population, and the random-directed actions performed by different operators during the search. Stochastic techniques such as GAs provide an alternative for the solution of large combinatorial problems as deterministic techniques cannot tackle these problems properly [GARE79].

   The stochastic nature of GAs may give rise to quite different computational behaviours, and this has been the principal motivation for advocating GAs for the solution of NP-complete problems [PROS91].

   Figure 3.4 illustrates this stochastic nature of GAs. Here, a GA has solved a problem 10 times and on each occasion, it has reached a different solution. This graph shows the reduction in terms of cost over 100 iterations for each of the 10 runs. The shape of these curves is typical, most of the improvements come at the start, and the end of the run only sees small improvements - or no improvement at all. This is a common situation for most optimisation algorithms.

2. Typically, search techniques move from point to point in the search space; instead, GAs work with a population of solutions. For this reason, a GA can be said to have a more global view of the search space than a conventional algorithm and is less likely to be trapped in some inferior region of the search space and, as a consequence, it is less likely to converge towards a poor local optimum [GREF87].

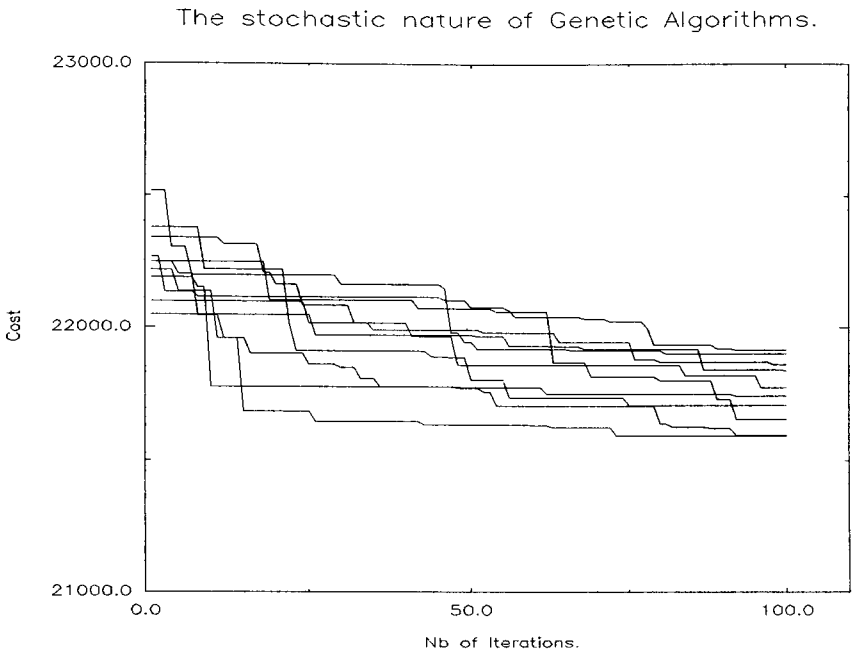

Figure 3.4: The stochastic nature of GAs.

3. The two previous features are integrated into an iterative process. Generation after generation, using random-directed decisions, a GA first selects above-average solutions for reproduction and then recombines them to produce a new set of solutions. If a piece of information is present in many above-average solutions, it will to be propagated to future generations. As stated previously, the ultimate objective of this iterative process is to "build" the optimal solution from the information present in the pool of solutions.

The following subsection introduces the theory behind GAs.

3.2.3 Genetic algorithms: the theory.

This subsection presents the schema (pl: schemata) theorem which provides a formal platform for GAs and can be used to explain their internal mechanisms.

What is a schema?

A schema is the generalisation or abstraction of a binary string; each element of a schema can take the values: 0,1 or #, where # means that the value is irrelevant (0 or 1). For instance, the strings "1000" or "0000" both contain the schema "#000".

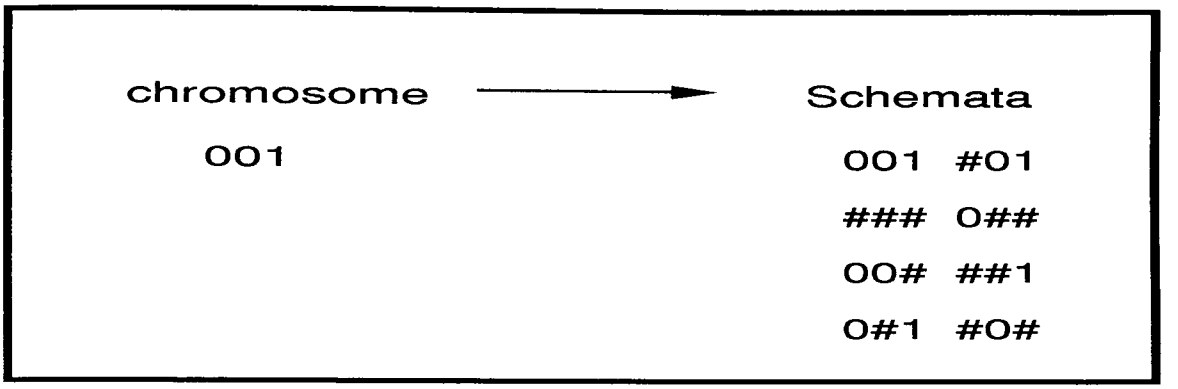

Figure 3.5: A chromosome and its schemata.

Figure 3.5 shows a chromosome and its schemata. A chromosome of length L contains 2^L schemata. It means that the evaluation of one chromosome implies the evaluation of 2^L schemata. At each generation, a GA recombines the schemata present in the different chromosomes of its population.

The schema theorem presented in the next paragraph demonstrates that the strength of GAs lies in the efficient processing of these schemata [DAVI91a].

The Schema Theorem.

The Schema Theorem [HOLL75] states that *"Assuming that selection chances are proportional to chromosome fitness, a schema occurring in chromosomes with above-average fitness will tend to occur more frequently in the next generation, and one occurring in chromosomes with below-average fitness will tend to occur less frequently (ignoring the disruptive effects of crossover and mutation)"*. This theorem immediately provides some answers to some important questions:

- What are the basic assumptions behind GAs?
- Why does a GA work with a population of solutions?
- What is the importance of one string?

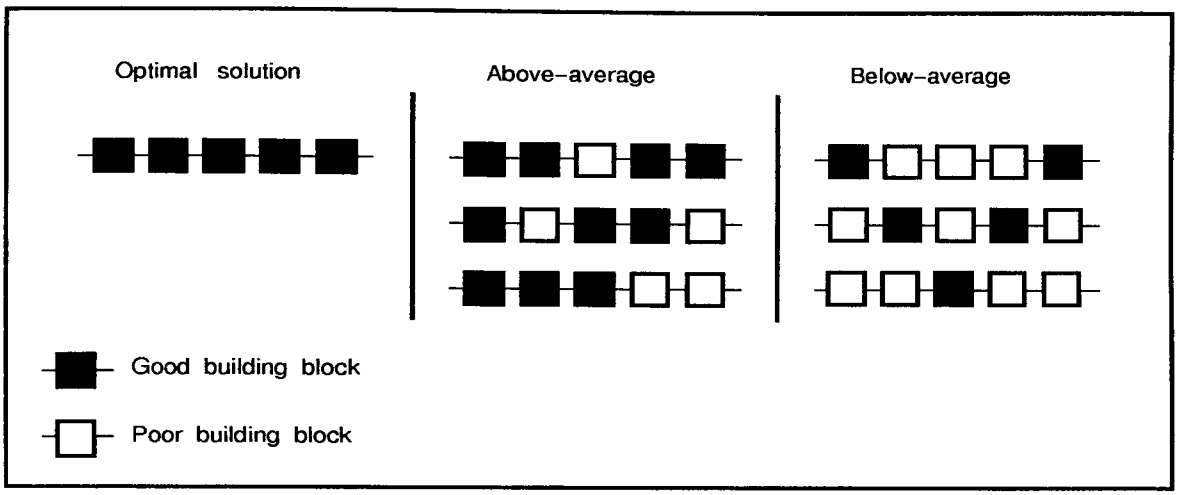

Figure 3.6: Building blocks.

Assuming a suitable problem-domain, the basic assumptions are that:

1. the optimal solution and the above-average solutions of a problem have some similarities; they have some common schemata.
2. the recombination of schemata present in above-average solutions should lead to the identification of the optimal solution.

Schemata can be viewed as building blocks. Given the assumption that a solution consists of a series of these building blocks (see Figure 3.6), the difference between an above-average solution and the optimal solution is that the optimal solution has all the right building blocks whereas an above-average solution has some of these building blocks but also some poor building blocks. Therefore, if a building block is present in several above-average solutions, that building block is likely to be part of the optimal solution.

In order to identify similarities among above-average chromosomes, a GA must work with a population of solutions. In fact, a chromosome by itself has no real importance. The search is guided by the similarities, i.e. if a schema is present in a large number of above-average chromosomes, this schema is likely to be propagated to the next generation. However, what happens to an individual solution, i.e. the chromosome, is not relevant [GREF87] [GLOV87].

3.2.4 The driving forces.

This subsection analyses a number of mechanisms and parameters which influence and guide the search process of a GA. As stated earlier, a GA has to (1) select above-average solutions for reproduction and (2) recombine them to produce a new set of solutions. Therefore, the different driving forces have two principal objectives:

1. Make sure that the recombination of similarities from above-average solutions is fruitful, i.e. the recombination of two above-average solutions tends to generate an above-average solution.

2. Maintain the best possible balance between solution quality and rapid convergence. Different applications may have different needs: time-critical applications (e.g. traffic, routing of telecommunications networks) will sacrifice optimality for a rapid convergence towards a satisfactory solution, whereas applications with no (or minimal) time-constraints (e.g. design and dimensioning applications) will favour optimality rather than rapid convergence.

Each driving force is now considered in turn:

Representation/operators.

The data structure used to represent candidate solutions, i.e. the representation, and the operators, i.e. crossover and mutation, cannot be considered separately; the operators must be able to work with the representation implemented and therefore, the choice of the data structure dictates the design of the operators. As a result, these two elements can be viewed as a united driving force which must ensure that:

1. Genetic operations are not disruptive[4] and also children should inherit a maximum of information from their parents.

2. The resolution of the representation is adequate. In fact, the representation can be seen as a discretisation of the search space. If the representation is too restrictive (i.e. over-constrained), some regions of the 'real' search space might not be accessible, while if the representation is too relaxed (i.e. under-constrained), the GA may be force to explore an unnecessary large search space. In both cases, the GA may not be able to converge towards the optimal solution.

The internal algorithms - crossover and mutation - used to encode the operators, are strongly dependent on the chromosomal representation adopted to solve a specific problem. However, these problem-specific algorithms perform similar functions, i.e. a crossover operator is always a crossover operator whatever representation is being used. The following paragraphs analyse these generic functions.

Crossover and mutation.

The roles of the crossover and mutation operators can be described as follows:

[4] A crossover operation is said to be disruptive if in general, the recombination of two chromosomes tend to generate a solution with a worse fitness than its parents.

The crossover operator acts by combining parts of two parents. Clearly, it recombines groups of bits from two chromosomes and produces two new chromosomes. It does not create any new genetic material but simply manipulates the information already existing in the population. It is analogous to mating in biological organisms.

Mutation operates on a single parent by randomly changing some part of it. It is a background operator, i.e. it occurs rarely, and the main task of mutation is to re-introduce diversity into the population during the search process in order to avoid premature convergence[5]. In genetic terms, this operation is analogous to the mutation of genes, in which the code for one amino acid changes to the code of a different amino acid [SING94].

Crossover is the key-operator for GAs, mutation alone could not generate the recombination process achieved by the crossover operator [SCHA92].

However, the actions of the crossover might be disruptive and these disruptions represent the main difficulty faced by GAs. The search performed by a GA is reduced to a basic random search if the children cannot inherit sufficient information from their parents and, therefore, the disruptions caused by the crossover must be kept to a reasonable level.

For certain classes of problems, it might not be possible to design a suitable crossover operator. For instance, crossover might be especially disruptive for problems with a high level of epistasis [6] where the various elements of the chromosome are severely constraining each other; the recombination of two above-average solutions is not likely to produce an above-average solution. Davidor advocates pure random search for a solution to these problems [DAVI91a].

[5]Premature convergence can be defined as the loss of diversity in the population before the optimal solution has been found.

[6]In this context, the term "epistasis" is related to the level of dependence existing among the different elements of a chromosome. If many elements are heavily dependent on the values assigned to other elements, the level of epistasis is said to be high.

It is also worth mentioning that GAs might not be the appropriate tool for the solution of problems with a low level of epistasis. In this case, the various elements of a chromosome are not constraining each other; the values they are assigned are more or less independent from each other and a hill-climbing procedure, focusing on each individual element separately, will be preferred. It is simpler to implement and, in this case, more efficient.

The initial population.

The initial population must supply a suitable sample of the search space to the genetic algorithm. The diversity of the initial population must be sufficient so that there is sufficient variety in the schemata to be processed.

In order to give a smart start to the GA, the initial population can be produced with a heuristic rather than using a random method. However, Grefenstette, in [GREF87], reports that the seeding of the initial population with above-average solutions generated by a heuristic can lead to a premature convergence. The author concludes that this heuristic-seeding must be done with care. Our results confirm this point (see Subsection 6.1.2).

The size of the population can also affect the efficiency of GAs. Very small populations may not create enough variety whereas large populations may be prohibitive in terms of computational effort.

The selection function.

The selection function guides the search by determining which chromosomes are used during the reproduction process; it also has a key role to play when deciding between optimality and running time.

For (near) real time applications where a rapid solution is desired, Davidor advo-

cates an exponential selection function: the best solutions are given an exponential chance of being selected. This strategy leads to a rapid convergence towards a good, but probably sub-optimal solution [DAVI91a].

For applications with no time-constraints, the primary concerns of the selection function are (1) to guide the search efficiently and also (2) to avoid a premature loss of diversity in the population [DAVI91b]. In particular, two critical situations might emerge and the selection function must handle them.

1. One chromosome, i.e. a super individual, might have a much better cost than the other members of the population. In this case, the selection function has to attenuate the selection chance of this particular chromosome. If the probability of being selected is directly related to the fitness of the chromosomes, it is likely that the super-individual will always be selected; the population will then lose its diversity and a rapid convergence towards a local optimum will happen. One possible solution is to normalise the situation and to make the chance of selection of each chromosome directly proportional to its ranking in the population, rather than to its individual fitness. This normalisation technique limits the risk of domination by one super-individual.

2. All the chromosomes have a similar fitness. In this case, chromosomes with the highest fitness must be given a better chance. In fact, in order to guide the search efficiently, the selection function must introduce some pressure and exaggerate the selection chances of the fittest chromosomes.

In summary, this subsection has discussed the importance of the different driving forces. These different ingredients all have a critical part to play in the success or failure of a GA. Firstly, assuming a suitable problem-domain, a representation and its associated operators must be designed so that:

- the action of the crossover creates as little disruption as possible and tends

to generate better solutions from one generation to the other,

- the resolution of the representation is adequate. It is necessary for the representation to provide a correct and complete view of the search space.

Moreover, the operators, the initial population and the selection function must be designed so that the best possible balance between optimality and running time is kept.

The remainder of this section on GAs contains three subsections describing the applications of GAs to the following problem-domains:

- Resource allocation (Subsection 3.2.5),
- Vehicle routing (Subsection 3.2.6),
- Telecommunications (Subsection 3.2.7).

3.2.5 GAs for resource allocation.

This subsection reviews a number of applications which use GAs for the solution of resource allocation, i.e. scheduling, problems.

The chromosomal representation that these systems use influences the design of the entire GA, that is, the design of the operators and the resolution of the search space. Therefore, these applications are reviewed with respect to how they represent the problems they try to solve and also, we present a study of the alternative representation schemes, and comment on their effectiveness, or otherwise.

Figure 3.7 presents a number of possible representations which can be divided into two broad classes:

- indirect representation,

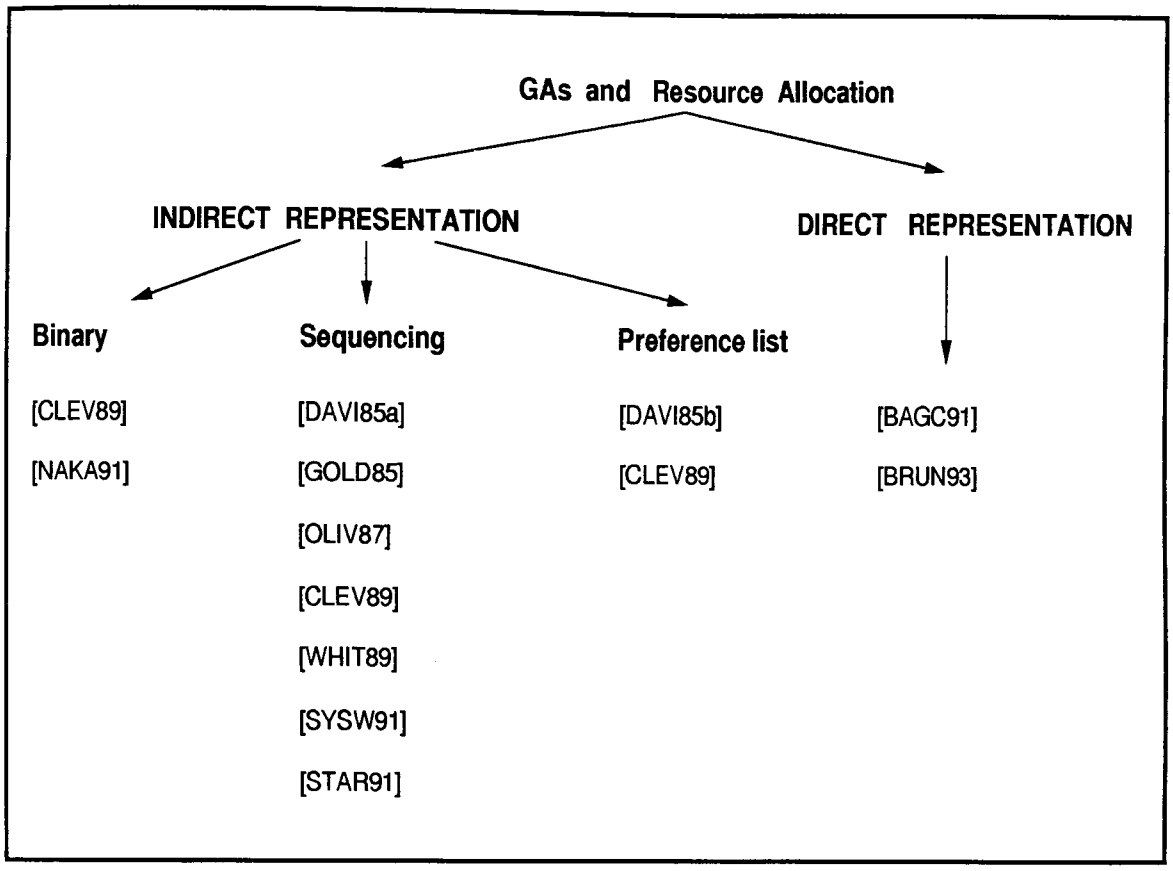

Figure 3.7: Representations for Resource Allocation Problems.

- and direct representation.

The next two paragraphs examine these two styles of representation.

The indirect representation.

The indirect representation is the most widely used. Davis was the first to propose the two-fold "GA-solution builder[7]" architecture [DAVI85a]. His claims were that for certain classes of epistatic problems such as bin-packing and graph colouring, the conventional, i.e. binary, representation is not suitable. The recombination of two above-average solutions is not likely to produce above-average solutions, rather

[7]or schedule-builder when solving scheduling problems.

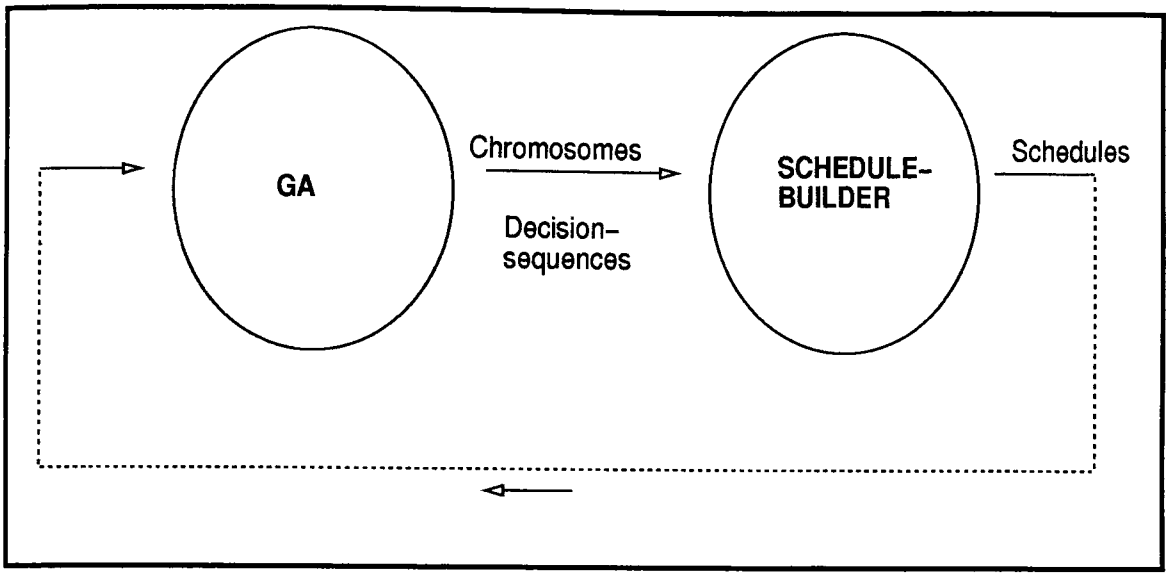

Figure 3.8: Indirect representation.

it is (highly) likely to produce illegal solutions. He, thus, presented an alternative technique, where the GA was only doing part of the search and a solution-builder was decoding the intermediate solutions manipulated by the GA.

Figure 3.8 illustrates how GA-based systems, using this representation, work.

The search process is shared by a GA and a schedule-builder. The GA uses an intermediate representation for its chromosomes. These chromosomes are viewed as decision sequences by the schedule-builder which uses them to generate legal solutions. Thus, each chromosome is associated with a legal schedule and the fitness of the chromosome is defined by the quality of the schedule.

Three different classes of indirect representations can be distinguished:

- binary representation,
- sequencing representation,
- and preference list.

These three sub-classes are now considered in turn.

Binary representation. Nakano applied a conventional GA to three job-shop problems; the largest problem had 20 jobs and 5 machines [NAKA91]. There were some sequencing constraints, and each job had to be processed on different machines following a given sequence.

Each bit of the chromosome was associated with a pair of jobs and a machine; the value of the bit (0 or 1) defined the processing priority between the two jobs on that particular machine.

As the search space covered by the binary representation was much larger than the real search space, most of the solutions produced by the crossover operator were illegal. As a consequence, a schedule-builder was needed to generate legal schedules. Nakano claimed that the solution-quality achieved by the GA approached those obtained by branch and bound methods. However, it is not clear what information is transmitted from generation to generation and how children can inherit that valuable information from their parents.

Cleveland and Smith compared a binary representation with some sequencing and preference list representations on a series of problems on scheduling flow-shop releases [CLEV89]. Their results are discussed later (see paragraph on preference lists).

Sequencing representations. This approach has been the most popular to date with the GA community for the solution of scheduling problems. Here, the problem is reduced to a sequencing problem. Basically, the original problem is transformed into the problem of finding an ordering (or sequence) such that the schedule generated from this ordering has a minimal cost.

Table 3.2 presents a collection of crossover operators which work with a sequencing representation:

The principal difference between these operators is the information that each of

Type	Order-based	PMX (see below)	Cycle crossover	Edge recombination	Position-based
Reference	[DAVI85a]	[GOLD85]	[OLIV87]	[WHIT89]	[SYSW91]
Pressure(s)	1. Order 2. Position	1. Order 2. Position	1. Position	1. Adjacency	1. Position

Table 3.2: Operators for the sequencing representation.

them attempts to preserve during the reproduction operation. Under "pressure" is identified the objective(s) for each operator. These are amplified below.

The GA developed by Davis [DAVI85a] was the first to integrate a sequencing representation and an order-based crossover. In this case, the objective of the crossover was to preserve the relative order of the elements contained in the sequences. During crossover, one part of the sequence is inherited from one parent and the other part is filled with the remaining elements, following the relative order of the other parent.

The partially mapped crossover operator (PMX) has been proposed in [GOLD85]; this operator is an order-based crossover which also attempts to preserve the absolute positions for one part of the child and the relative order for the other part. However, Starweather et al. claimed that the PMX operator is more disruptive than the order-based crossover [STAR91].

The cycle crossover has been designed by Oliver [OLIV87]. The objective of this operator is to preserve positional information. Each position of the sequence is guaranteed to inherit the value assigned to the same position in one of the two parents.

The edge-recombination operator was specifically designed for the solution of TSPs [WHIT89]. The authors claimed that the important information, i.e. the information that children must inherit from their parents, for this particular class of problems, is not the absolute positions or the relative order of the elements in the

sequence. Rather, it is the fact that some elements are next to each other in the sequence. Basically, the important information in a solution of a TSP is the fact that city a is visited just before or just after city b, i.e. the fact that cities a and b are adjacent in the tour. Hence, preservation of adjacency information is the objective for this operator.

The position-based operator [SYSW91] had the same objective as the cycle crossover; it attempts to preserve positional information. However, Starkweather et al. claimed that this operator is more disruptive than the cycle crossover and is in fact a variant of the order-based crossover [STAR91].

The performances of these different operators vary from one problem-domain to another, and are, in fact, strongly dependent on the specific nature of the problem in hand. For instance, in [OLIV87], the authors compared the performance of three operators: the order-based crossover, the PMX and the cycle crossover on a single 30 cities TSP. The order-based crossover did 11% better than the PMX and 15% better than the cycle crossover. They also compared (1) the tours achieved by a GA with an order-based crossover and (2) the optimal solution; the best results were within 0.25% of the optimal solution.

Syswerda, in [SYSW91], compared three crossovers (order-based, position-based and edge recombination) on a job-sequencing problem with resource, time and set-up time constraints, and with 90 tasks to be scheduled on 30 resources. The performances of the position-based and the order-based crossover mechanisms were quite similar whereas the edge recombination operator produced rather poor results and was, in fact, doing little better than random search. This operator, designed especially for the solution of TSPs, was obviously not suitable for this type of scheduling problems.

Starkweather et al., in [STAR91], compared six crossover operators on two different problems: a 30 city TSP and a real world warehouse/shipping scheduling problem

with 195 jobs. The six operators were: the edge recombination crossover, two variants of the order-based crossover, the position crossover, the PMX and the cycle crossover (see Table 3.3).

	TSP	Scheduling problem
1	edge recombination	position-based
2	order-basec (1 and 2)	order-based (1)
3	position-based	PMX
4	PMX	order-based (2)
5	cycle	edge recombination
6	-	cycle

Table 3.3: Sequencing crossovers - A comparison.

For the TSP, the ranking of the operators was as follows: (1) edge recombination operator, (2) order-based crossovers, (3) position-based crossover, (4) PMX and (5) cycle crossover. The edge recombination, the two order-based crossovers and the position-based achieve similar results; they were all within 1 % of the optimal solution. Compared to the previous operators, both the PMX and the cycle crossover performed poorly on this problem and they were respectively 10% and 20% away from the optimal solution.

For the second problem, the scheduling problem, the ranking of the operators was rather different. The edge recombination operator performed poorly compared to its competitors. The position-based crossover and one of the two order-based crossovers achieved the best result, followed by the PMX operator and the remaining order-based crossover. The cycle crossover was again quite disappointing.

In summary, all these comparative studies show how much the performance of the different operators depends on the specific nature of the problem in hand. The edge recombination operator attempts to preserve adjacency and, thus, is suitable for the solution of TSPs; however, it might not be the best choice for scheduling problems where the notion of adjacency is not so important. The order-

based crossover and the position-based crossover appear to be a suitable choice, for instance, for problems with precedence constraints where the relative order might be the relevant information. None of these studies has been favourable to the cycle crossover; however, this operator might be required to apply GA to sequencing problems where the absolute positions of the various elements in the sequence is of prime importance.

Preference lists. This representation, proposed by Davis [DAVI85b] for the solution of a small job shop scheduling problem is rather rich and complex. The schedule-builder is, in fact, a simulator which must take decisions for each machine at regular intervals; the decision can be to allocate a task to a machine or, the machine may have either to wait for a job or be idle for a given time period.

The chromosome (here, preference list) dictates, for each of these decision points, which decision the simulator should prefer. Clearly, for each decision point, the chromosome provides an ordered list of possible decisions; each of these is attempted by the simulator until one can be accomplished.

Davis [DAVI85b] designed three operators for this representation:

1. "Run-idle" attempted to limit the amount of waiting time experienced by the different machines.
2. "scramble" "scrambled" (or rearranged) the elements of a preference list.
3. "crossover" exchanged preference lists between solutions.

The principal limitation of this representation is that it is not a practical approach for large problems. The preference list must provide an ordered list for each decision-point and for each machine, and therefore can become very large very rapidly when the size of the problem increases.

Cleveland and Smith undertook a comparative study of binary representations, sequencing representations and preference lists [CLEV89]. Two complex single-machine scheduling problems were used as the basis for the comparison. In fact, the same problem was viewed from two different angles. In the first case, the scheduling problem was reduced to a sequencing problem where the order in which the jobs were released was the only concern. In the other case, the problem was viewed as a more general timing problem and, absolute release times for each job were required.

As expected, sequencing approaches performed much better than both the binary representation and the preference list on the sequencing problem. The principal difference between these three techniques was that both the binary representation and the preference list introduced the notion of time in their representation whereas the sequencing approaches were only concerned with the order of the jobs in the decision-sequences. The authors concluded that the binary representation and the preference list were, in a sense, too complex for this problem and could not guide the search in an efficient manner.

The sequencing approaches showed their weakness on the second timing problem. Indeed, in this problem where absolute release times were part of the solution, both the binary representation and the preference list provided significantly better results than the sequencing approaches.

In summary, three classes of indirect representations have been examined. Until recently, indirect representations were the most popular choice for the solution of scheduling problems. The principal reason for this was the relative simplicity of the operators involved.

Direct representations.

Direct representation has been proposed as a promising alternative for the solution of resource allocation problems [BAGC91] [BRUN93]. The genetic operators are more complex. However, they overcome most of the limitations encountered by their indirect counterparts, with which they differ in two principal ways:

1. The GA works directly on the original search space, i.e. the schedules are the chromosomes. There are no distortions introduced by an intermediate search space.

2. The schedule builder is eliminated. More precisely, the scheduling function is performed by the crossover operator. The GA performs as much of the search as possible. Basically, all the variables are encoded in the representation and hence, the schedule builder might not be needed at all or used at worst to refine marginally the solutions proposed by the recombination process.

The definition of suitable operators, crossover and mutation, represent the principal problems with this class of representation. In comparison with the sequencing representations, where the crossover operator only recombines two sequences of elements, the task is here much more complex. The crossover must mix and recombine two schedules and guarantee that, during this operation, the schedules have inherited the important information from their parents.

Bagchi et al., in [BAGC91], used a job shop scheduling problem to compare three different representations; the principal difference between these three representations being the amount of information encoded in the representation.

For the simplest representation, the original problem was reduced to a sequencing problem and the chromosomes were lists, i.e. permutations, of jobs. The schedule-builder had an important role to play in the search process. At the other end of

the spectrum, the third representation integrated all variables and hence, the entire search process was performed by the GA.

In this comparative study, the performance of the GA was directly related to the amount of knowledge encoded in the representation; the more information that was encoded the better was the performance.

In [BRUN93], the author also puts forward a strong case for the direct approach. A large real-world problem was used to compare one sequencing representation and one direct representation. Experiments were performed with different settings, i.e. different sizes of populations, initialisation heuristics, number of iterations, selection methods. The direct approach with its "knowledge-augmented" representation consistently outperformed the sequencing approach.

In conclusion, this section has presented two styles of representations, indirect and direct. Until recently, the indirect style was favoured because of the simplicity of its operators. However, direct approaches are now emerging. They do represent a promising alternative to the more traditional indirect approaches and merit further investigation.

3.2.6 GAs for Vehicle Routing.

This subsection reviews a number of applications which use GAs to solve VRPs. All of the systems presented in this section use an indirect representation; each of these systems is now presented in turn:

Thangiah et al.[THAN91] developed *Gideon* which addressed VRPs with time windows and capacity constraints; the objective of this system was to "minimise the number of vehicles and the distance travelled for servicing the set of customers without being tardy or exceeding the capacity or travel time of the vehicles".

In *Gideon*, a three-stage strategy was adopted. In a first step, the region covered by the different vehicles was divided into sectors; all the customers located in one sector were serviced by the same vehicle. Then, a heuristic was used to build the routes of the different engineers, i.e. given a cluster of jobs, the heuristic was used to find the best possible route among these customers. Finally, a local search (improvement) procedure was employed to refine the final answer.

Only the first step of the search involved a GA; each chromosome was a binary string which represented a list of seed angles used to divide customers into clusters. The objective of the GA was to find a set of seed angles minimising the total distance travelled by the engineers.

The authors reported two comparative studies of *Gideon* with other techniques. In [THAN91] , *Gideon* was compared to two local procedures developed by Solomon [SOLO83]. Three different classes of VRPs were used:

- Class **I**: VRP where locations of customers were generated randomly,
- Class **II**: VRPs with clusters of customers,
- Class **III**: VRPs with semi-clusters of customers.

These problems also varied in terms of fleet size, vehicle capacity, travel times, time-windows, and customer service times. In 41 out of 56 problems, *Gideon* performed better than the local search procedures; the average reduction in terms of fleet size was 3.9% and in terms of distance travelled by the vehicles, the average reduction was 4.4%. In general, the improvements achieved by the GA were greater for problems in Classes I and III than for problems in Class II.

In [THAN93], a set of 25 large problems (200 customers for each problem) was used to compare the performance of *Gideon* with the performance of a greedy algorithm. Again, the geographical distribution of the customers was used as a parameter to divide the VRPs into 5 categories, each containing 5 problems.

In the first class, the customers were uniformly distributed across the entire area, while in the other four classes the 200 customers were regrouped respectively into 1,2,3 or 4 clusters. Within each class, the problems varied in terms of time deadlines, and varied from loose to tight constraints.

In this comparative study, three sets of experiments were performed. In the first set, the different problems were solved solely by using the greedy algorithm. In the second set, the greedy algorithm was augmented by a post-optimisation procedure and finally, *Gideon* was used in the third set of experiments.

In terms of solution quality, *Gideon* performed well for problems where the customers were distributed uniformly and/or with tight time deadline constraints. On the other hand, the association, greedy algorithm and post-optimisation, achieved good results for problems in which customers were tightly clustered and with loose time deadlines.

In terms of computational effort, the comparison is more delicate. In most cases, both versions of the greedy algorithm, i.e. with and without the post-optimisation, outperformed *Gideon*. However, in some cases, the greedy algorithm augmented by the post-optimisation procedure was extremely slow in providing an answer.

Blanton and Wainwright also used a GA to solve VRPs with time-windows and capacity constraints [BLAN93]. Their principal claim was that traditional operators usually associated with the sequencing representation (e.g. order-based, position-based, cycle crossovers) were not suitable for the solution of problems with multiple constraints. In order to overcome this limitation, a new class of crossovers was developed, namely the merge cross operators, which use global precedence, i.e. priority, information during the recombination process.

Basically, a global precedence arrangement indicates, for each pair of customers, the order in which the customers should be processed. Different criteria (e.g. time-windows of the different customers, distances or capacities) might be used to gen-

erate different arrangement and hence, different crossovers.

Twelve merge cross operators, using different arrangement criterion (or combination of criterion) were compared with three traditional crossovers, PMX, cycle crossover and the edge recombination operator, on four different problems with 15, 30, 75 and 99 customers respectively. In each case, the best solution was achieved by one of the merge cross operators. As a group, these operators outperformed the traditional operators. However, no merge cross operator was consistently better than the rest of its competitors. This raises some serious doubts concerning the robustness of these operators. In fact, the success or failure of a given merge cross operator appears to be greatly dependent on the specific characteristics of the problem in hand.

All the systems presented in this subsection used an indirect representation. No GA with a direct representation has yet been implemented to solve VRPs. The comparison in Section 3.2.5, where systems developed by Bagchi et al. and Bruns outperformed their competitors, motivates the investigation of this class of representation for the solution of VRPs.

3.2.7 GAs for telecommunications.

This short section presents three recent applications of GAs in the field of telecommunications.

Many features of typical telecommunications problems motivates a GA solution. Indeed, a large number of telecommunications applications have large and complex search spaces and/or have severe time-constraints. In both cases, conventional techniques might fail to provide an answer in reasonable time, and non-deterministic search techniques such as GAs may be well-suited for the solution of these problems.

Clitherow and Fisher, from Bellcore, developed a GA to solve small design problems

(from 5 to 30 nodes) [CLIT89]. In fact, two GAs were implemented. The first GA used conventional operators whereas the second was equipped with problem-specific operators. These problem-specific operators were two mutation mechanisms whose actions were dictated by a rule base, a set of heuristics used usually by human experts to design a network manually.

In both GAs, each element of the chromosome represented the capacity of a particular link between two nodes, i.e. there was one element for each link of the network.

The GAs were given a smart start; the initial population was not purely random. For each element of the chromosome, the probability distribution used to define the initial capacity was centred around an approximate solution.

The principal results of the study were that:

1. The GA equipped with the two problem-specific mutation operators consistently outperformed the conventional GA,

2. The number of iterations required for convergence increased as the network grew in size; hardly a surprising outcome !

Cox proposed an algorithm for the dynamic anticipatory routing (DAR) of calls in circuit-switched telecommunications networks [COX91]; this scheduling problem was concerned with the capacity reservation for services such as video broadcasting where demand can be planned in advance.

The elements of the problem were:

1. The configuration of the network, e.g. number of nodes, links and capacities,

2. The traffic pattern which was described in probabilistic terms,

3. A call table describing the individual calls requested (e.g. source node, destination node, desired start time, duration)

4. A loss function defining a penalty cost for each call type.

The objective was to find a dynamic routing policy minimising the average loss per unit of time resulting from blocked calls.

The DAR algorithm incorporated a two-fold procedure:

1. In the first part, the algorithm was using the traffic pattern only to solve the problem in a probabilistic manner and provide a path assignment rule to each call. This path assignment rule was, in fact, a small set of path selection probabilities mapping call i into path j with probability $p(i,j)$. The objective was to minimize the expected penalty cost, generated by blocked calls, per unit time.

2. The objective of the second part, called the bandwidth packing algorithm, was to improve the initial assignments made in the first part, by using information relevant to the current situation: (a) the current state of the network and (b) a list of call requests with their characteristics.

Four different algorithms, namely a greedy algorithm, a pairwise exchange heuristic, uniform random search, and a GA were implemented to perform the bandwidth-packing task. The performance of the four algorithms was compared using a 50 node test problem. All perform reasonably well. The worst one, uniform random search, was only 8% away from the optimal solution. In comparison to its competitors, the GA performed well, but was outperformed by a hybrid algorithm: GA followed by the pairwise exchange procedure.

Finally, Liepins and Potter reported a GA approach for diagnosing multiple faults in a microwave communications system [LIEP91]. The authors defined multiple

diagnosis as the identification of a set of faults that best corresponds to a set of alarms. First, the original diagnosis problem was mapped into a set covering problem and the GA was used to find the optimal answer to this constrained problem.

In conclusion, two important observations can be made from the reviews presented here:

1. The performances achieved by the different GAs, presented in the three previous sections, compared generally favourably with those achieved by their competitors.

2. Subsection 3.2.5, *"GAs for resource allocation"*, has highlighted the reason why a direct representation is an appropriate choice. This style of representation should be investigated for the solution of VRPs.

3.3 Simulated Annealing.

3.3.1 Introduction.

SA is an iterative and stochastic search procedure which was first introduced by Kirkpatrick et al. [KIRK83] and Cerny [CERN85]. The principal characteristic of SA is that it provides a mechanism which (potentially) prevents a local search algorithm from being trapped in a local minimum.

Local search algorithms are often used to solve large combinatorial problems for which exact algorithms do not exist or cannot provide an answer in reasonable time.

Usually, a local search algorithm works as follows: the algorithm starts with an initial solution which might be either chosen at random or provided by another search procedure.

A rearrangement operator is used to generate a neighbour [8] of the current solution; a neighbour can be defined as a similar solution which differs (from the original solution) only by a single change which is performed by the rearrangement operator.

Then, the difference of quality between the current solution and its newly generated neighbour is calculated. If the neighbour is of better quality, it becomes the new current solution. If not, the original current solution is kept. Thus, a local search algorithm can be viewed as a descent algorithm. Only moves which decrease the cost, i.e. downhill moves, are accepted. Moves which increase the cost, i.e. uphill moves, are always rejected.

These two steps of neighbour generation and comparison are repeated until no improvement can be detected in the neighbourhood of the current solution. Thus, this final solution is guaranteed to be a local minimum.

However, this local minimum might be distant from the optimal solution. In fact, the principal limitation of local search algorithms is that they do not provide a mechanism which would allow an escape from local minima. If the algorithm has reached a local minimum, the search can go no further. There is no exception to this rule, even if the local minimum found is of poor quality in global terms.

A simple answer to this problem is to repeat the local search from different starting points and keep the best solution found. In this case, different parts of the search space are explored and, intuitively, the algorithm is more likely to find a final solution of better quality. However, this naive approach does not always overcome the underlying problem.

SA is an alternative approach. Basically, this procedure accepts uphill moves with a controlled probability and is therefore less likely than a local search algorithm to be trapped in a poor local minimum.

[8] and therefore, the neighbourhood is the set of all the neighbours.

SA has been used to tackle some classical combinatorial problems, e.g. travelling salesman [GOLD86], graph partitioning [JOHN89b], graph colouring problems [JOHN91] and has given some mixed results.

The remainder of this section consists of five subsections. Subsection 3.3.2 introduces SA and describes its internal algorithm. Subsection 3.3.3 examines the role played by the different parameters. Subsection 3.3.4 reports the results of some comparative studies of SA with conventional techniques when applied to different classes of combinatorial problems.

3.3.2 SA: A brief description.

SA originated from an analogy drawn between thermodynamics and combinatorial optimisation. In this context, the search for an optimal solution in a combinatorial optimisation problem is analogous to the production of a crystal using an annealing technique.

The annealing technique works as follows: first, a solid is melted at a high temperature and then, the temperature is slowly and gradually lowered with most of the time spent at temperatures near freezing.

At high temperatures, the atoms of the system can move about freely. In the ground states, the particles are arranged in a highly structured manner and the energy of the system is minimal; the annealing technique has produced a crystal.

These ground states and this pure crystal can only be obtained if the temperature is reduced in slow stages. Otherwise, if the cooling process is too quick[9], the system will not reach its minimum energy state and the crystal will have many defects and only locally optimal structures. Basically, the period of time at each temperature

[9]The process in which the temperature is instantaneously lowered is defined as quenching. In the context of simulated annealing, local search is analogous to quenching.

must be sufficiently long to reach equilibrium.

The Metropolis algorithm [METR53] developed for the simulation of a collection of atoms in equilibrium at a given temperature is the central part of a SA algorithm. This algorithm works as follows:

- An atom is given a random displacement and the resulting change in the energy level of the system, ΔE, is calculated.
- If ΔE is negative, the newly generated configuration is accepted as the new configuration.
- If ΔE is positive, the case is treated in a probabilistic manner. The probability of acceptance is given by:

$$P(\Delta E) = exp(-\Delta E/K_b T)$$

  where K_b is the Boltzmann constant and T is the temperature.
- If $P(\Delta E)$ is greater than a random number between 0 and 1 then the newly generated configuration is accepted, otherwise it is rejected and the original configuration remains the current configuration.

In summary, the Metropolis algorithm allows both uphill and downhill moves to occur. However, there is an important difference: a downhill move is always accepted, whereas an uphill move is only accepted with a certain probability which decreases with the temperature, i.e. the lower the temperature, the less likely an uphill move will occur.

The SA algorithm is illustrated in pseudo-code in Figure 3.9. The inner loop of this algorithm is a Metropolis algorithm and the outer loop determines the changes in temperature. A SA algorithm works as follows:

```
1    S := initial-solution;
2    T := initial-temperature;
3    WHILE (not frozen)
4    DO BEGIN
5        WHILE (not equilibrium)
6        DO BEGIN
7            S' := nearby-solution(S);
8            Δ := change-in-cost(S, S');
9            IF (Δ > 0)
10           THEN accept S' with probability exp(−Δ/T) as S
11           ELSE accept S' as S
12           END
13       T := reduce-temperature(T);
14       END
15   END;
```

Figure 3.9: Implementation of simulated annealing.

First, an initial solution and an initial temperature are chosen; this initial temperature is set high, so that the term $exp(-\Delta/T)$ [10] tends to be large and large uphill moves are allowed.

The SA algorithm attempts a certain number of moves at each temperature and the Metropolis algorithm is used to define the validity of these moves. At each temperature, the algorithm must proceed long enough for the solution to reach a state of equilibrium.

The temperature is gradually dropped, and as T decreases, the term $exp(-\Delta/T)$ decreases, minimising uphill changes to the system. Eventually, as T tends to zero, $exp(-\Delta/T)$ also tends to zero, and no uphill moves can take place, i.e. freezing occurs.

[10] Note that T is a control parameter using the same units than the cost function and therefore, the Boltzmann constant K_b is not required anymore.

3.3.3 The driving forces.

This subsection analyses the different parameters which guide and influence the search performed by a SA algorithm. These parameters are:

1. A neighbourhood and a set of operators which define (1) the search space the algorithm has to explore and (2) how a neighbour solution can be generated.
2. An initial temperature selected so that it allows most uphill moves.
3. A temperature function which defines how the temperature is reduced during the search. The rate of reduction in T affects the granularity of the schedule. If T decreases too fast then quenching would occurs and it would generate a non-optimal solution (i.e. a glass instead of the desired crystal).
4. The number of iterations to be performed at each temperature.

Neighbourhood/operators.

The design of a suitable neighbourhood is critical to the success or failure of SA. Eglese [EGLE90] claims that a search space with a "smooth" topology where the local optima are shallow should be preferred to a search space with a "bumpy" topology where there are many deep local minima. Eglese, in the same paper, also covered constrained problems. The two traditional technique for solving such problems were proposed: (1) restrict the search space to solutions which conform to all the constraints or (2) allow solutions which do not satisfy all constraints but which are then given a penalty. The author claimed that the second technique is likely to generate a simpler search space with a smoother topology.

The annealing schedule.

The initial temperature, the temperature function and the number of iterations performed at each temperature define the annealing schedule which is also called the cooling schedule. Eglese reported several classes of annealing schedules [EGLE90]:

1. The original annealing schedules [KIRK83]. The initial temperature is set high enough to allow most uphill moves. A proportional temperature function (i.e. $T(t+1) = \alpha T(t)$) is implemented and the decrements becomes smaller and smaller as t increases so that an important part of the search is performed at low temperatures. At a given temperature, equilibrium is reached and the algorithm moves to a lower temperature when a sufficient number of transitions have been accepted, subject to a constant upper bound. This original annealing schedule can be modified in several ways: for instance, the number of iterations performed at each temperature might be kept constant, or be proportional to the size of the problem in hand, and so on.

2. Hajek [HAJE88] proposed an important theoretical result, stating that, in order to reach an optimal solution, there was no requirement to attain equilibrium at a succession of reducing temperatures. Rather, the main condition was for the cooling to be carried out sufficiently slowly. This result meant that there is a balance between a large reduction in temperature and a small number of iterations at a fixed temperature. It also generated a spectrum of alternatives for an annealing schedule, with the two extremes being:

    - The annealing schedule proposed by Lundy and Mees [LUND86] where only one iteration is performed at each temperature and the following temperature function is used:

$$T(t+1) = T(t)/(1 + BT(t))$$

where B is a constant. This function provides a slower temperature reduction than the proportional function used by Kirkpatrick et al. [KIRK83].

- The annealing schedule proposed by Connolly [CONN88] where the temperature is kept constant during the search. This can be viewed as a random search which accepts uphill moves with the probability $exp(-C\Delta)$ and C is kept constant during the entire search process. Connolly used this method for the solution of quadratic assignment problems [11] (QAP). This work achieved results which outperformed the results achieved by the annealing schedule proposed by Kirkpatrick et al. [KIRK83]. The author also claimed that, for this class of problems, a sequential generation of neighbours was superior to a random generation.

3.3.4 Comparative studies.

This subsection reports the results of several papers which compared the performances of SA with the performance of other search procedures on different classes of combinatorial problems. For each paper, this study (1) presents the class of problems being examined and (2) examines and compares the results achieved by the SA algorithm and its competitors.

Golden and Skiscim used SA to solve routing and location problems, in particular travelling salesman problems (TSPs) and p-median location problems[12] [GOLD86].

A set of TSPs of varying sizes (between 25 and 318 cities) were used to compare the results achieved by SA[13] with those of:

[11]The quadratic assignment problem can be defined as the assignment of inter-communicating objects to locations such that the total cost of communicating is minimise.

[12]The p-median problem is to locate p facilities at various nodes of a network in such a way that the sum of the distances from each node in the network to its nearest facility is minimised.

[13]In these tests, the SA algorithm implemented used a 2-opt procedure as its main operator.

1. A specialised heuristic for the TSP; the CCAO procedure. The authors described the CCAO heuristic as a *"hybrid procedure that use the convex hull of points as the starting sub-tour and insert nodes using a combination of the greatest angle method and the cheapest insertion criteria."* Also, a branch exchange heuristic was used as a post-optimisation procedure.

2. The classic 2-opt procedure of Lin and Kernighan [LIN73].

SA was consistently outperformed by the CCAO procedure, both in terms of quality and running time, and could only provide better results than the 2-opt procedure on a single problem.

For the p-median problems, SA was compared with a local search procedure, specifically designed for this class of problems. In general, this procedure required considerably less computational effort than SA and provided results of comparable quality.

Golden and Skiscim also studied the relationship between problem size and running time and concluded that, in practice, SA required a computational effort which grows faster than cubically.

Johnson et al. used SA for the solution of graph partitioning problems and compared its performance with standard local search algorithms [JOHN89b]. The graph partitioning problem is the problem of partitioning the vertices of a graph into two equal size sets in order to minimise the number of edges with end points in both sets. The authors used two classes of graphs for their comparison: (1) random and (2) geometric graphs.

For random graphs, SA outperformed its competitors both in terms of quality and running time, particularly, when the density and/or the size of the graph increased. However, for geometric graphs, the situation was reversed and local search procedures performed better than SA. By way of summary, the authors

offered a few general observations from their extensive series of tests and it appeared that:

- SA required long running times,
- the best way to add time was to add it uniformly at each temperature,
- there was no gain spending much time at high temperatures,
- the temperature function provided by Kirkpatrick et al. [KIRK83] was as good as other techniques, e.g. logarithmic cooling, linear cooling,
- there may be an advantage in starting with a good solution rather than a randomly generated one,
- replacing $exp(-\Delta/T)$ by an approximation or a look-up table could provide an important speed-up without degrading the quality of the solutions achieved.

Finally, Johnson et al. also examined the performances of SA for the solution of graph colouring and number partitioning problems [JOHN91]. Given a graph and n colours, the graph colouring problem is to assign a colour to each node of the graph such that no pair of nodes connected by a link have the same colour. In the sequence partitioning problem a sequence of real numbers $a_1, a_1, ..., a_n$ in the interval $[0, 1]$ must be partitioned into two sets $A1$ and $A2$ such that:

$$\left| \sum_{a_i \in A1} a_i - \sum_{a_i \in A2} a_i \right|$$

is minimised.

For graph colouring problems, three implementations of SA, with different internal representations, were tested and in general, achieved good final solutions at the expense of long calculation times. The authors viewed number partitioning problems as a class of deceptive problems for local search heuristic, and hence, SA. The tests

confirmed their view and SA was outperformed by its competitors both in terms of solution quality and running time.

3.4 Tabu Search.

3.4.1 Introduction.

Tabu Search [14] (TS) is a neighbourhood search technique which was first developed by Glover [GLOV86] [GLOV88] [GLOV89b]. Like SA, TS can escape from a local optimum by accepting uphill moves but in contrast with SA (and GA for that matter), most implementations of TS are wholly deterministic; the major exception being the probabilistic Tabu Search proposed by Faigle and Kern [FAIG92] which selects moves based on probabilities. The term *tabu* refers to the fact that the algorithm prevents the occurrence of some moves during the search; these moves are *tabu*. The objectives of TS can be stated as follows:

- To avoid being trapped in a local optimum.
- To avoid cycling, i.e. avoid wasting computational effort by re-visiting points in the search space.
- To maintain a correct balance between intensification and diversification, i.e. balancing between focusing the search effort in a specific (promising) area of the search space and encouraging the exploration of new regions.

In order to achieve these aims, the algorithm simulates a number of memory modules with different time spans (short-, intermediate- and long-term). These modules

[14] This technique was originally developed in North America. This explains why it is called *tabu search* instead of *taboo search*.

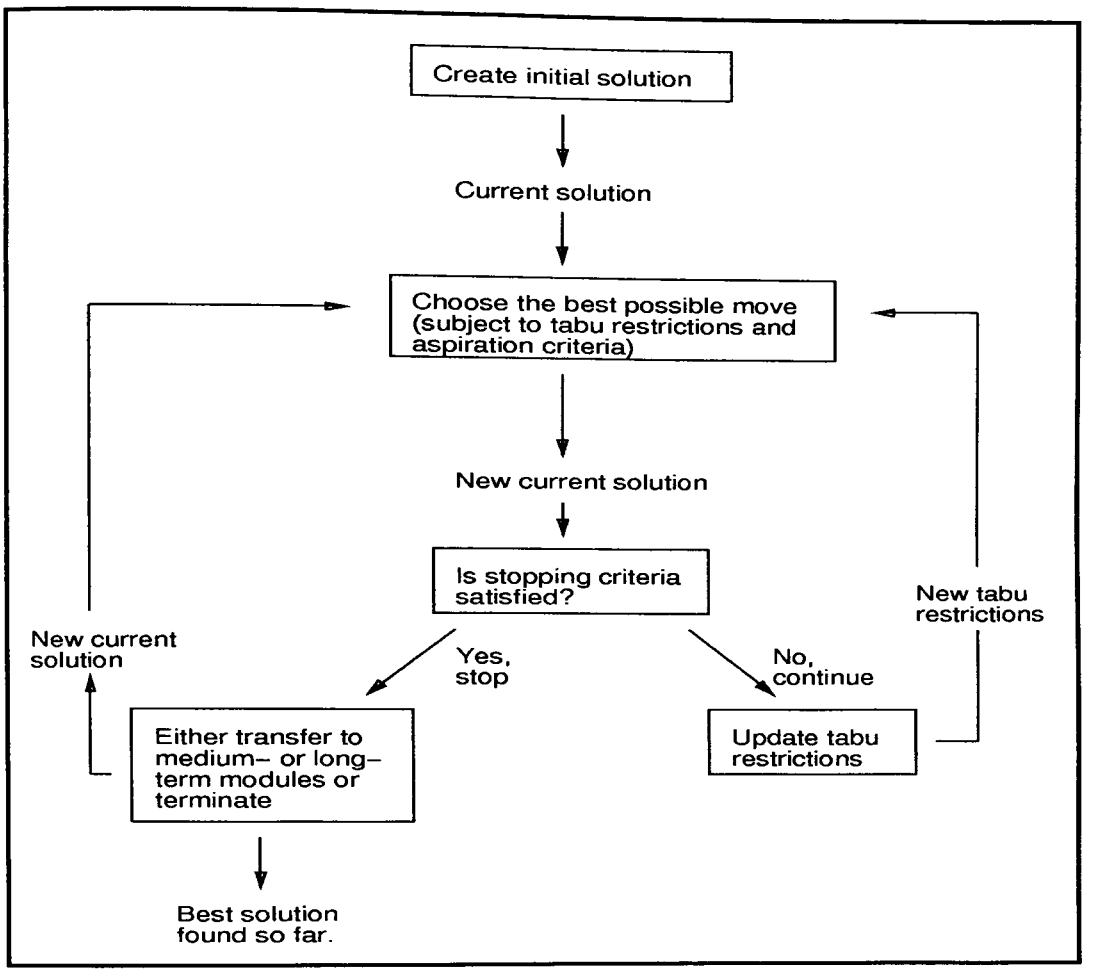

Figure 3.10: Tabu search - Short-term memory module.

constrain the search in different ways, and their interactions provide the means for answering the above objectives.

The remainder of this section consists of 4 subsections. In Subsection 3.4.2, the central part of TS, i.e. the short-term memory module, is described step by step. Subsection 3.4.3 examines the roles played by the different driving forces of TS while Subsection 3.4.4 reviews a number of applications.

3.4.2 TS: a brief description.

TS can be implemented in many ways and this paragraph describes the central part of most implementations (see Figure 3.10); this module corresponds to the short-term memory of TS. Given an initial solution S, the algorithm explores the entire neighbourhood $N(S)$ of S and consider all possible moves in both uphill and downhill direction. All moves are considered except for a certain number of tabu moves (tabu moves which meet the aspiration criteria [15] are also considered). The algorithm then continues its search by moving to S' (and S' becomes S); S' being the solution with the highest evaluation in the neighbourhood even if it implies an uphill move. With this mechanism [16], TS avoids being trapped; it does not terminates when it reaches a local optimum.

This cycle "exploration, selection" is repeated until a certain termination criterion is satisfied. At this point and depending on the implementation, either the algorithm terminates or the solution is passed on to the medium- and long-term memory modules.

3.4.3 The driving forces.

Tabu Search is composed of three modules:

- a short-term memory module which, at each iteration, seeks to make the best possible move. This module is constrained by the tabu restrictions and the aspiration criteria (this module was described in some detail in the previous section).
- a medium-term memory module which intensifies the search.

[15] Basically, a move is admissible either if it is not tabu or if its tabu status can be overridden by the aspiration criteria.

[16] With this mechanism, at the end of the search, the current solution S might not be the best solution encountered during the search. In order to overcome this problem, TS keeps a record of its best solution.

- a long-term memory module which diversifies the search.

Each driving force, namely the tabu restrictions, the aspiration criteria, the medium-term memory and the long-term memory modules, are now considered in turn:

Tabu restrictions.

In order to avoid being trapped in a local optimum, TS allows non-improving moves. However, this means that the algorithm could cycle around the same solutions [17] unnecessarily. The aim of the tabu restrictions is to prevent this cycling behaviour. The tabu restrictions can be implemented as follows: each time a move is made, its "inverse [18]" is added to a FIFO (First In - First Out) list called the tabu list. This list has a maximum length of T (called the tabu tenure). This limit means that a move only remains tabu for a limited amount of time [19].

In order to avoid cycling, the ideal situation would be to record each solution already visited during the past search and to forbid the algorithm to re-visit these points. Unfortunately, this approach is not acceptable for large-scale applications. For such applications, the storage of all previous solutions would require a prohibitive amount of memory and the cost of comparing the current solution against (potentially) all previous solutions would also be prohibitive. This explains why TS only records attributes of a limited number of past solutions.

It should be noted that an algorithm may have several tabu lists, with each list dealing with a specific set of attributes. In this case, Glover recommends that each tabu tenure (i.e. lengths of the tabu lists) should be different. His point is that,

[17] For instance, the following situation may arise, the algorithm takes a non-improving move to escape from a local optimum, only to return to that point in the very next move because the algorithm is designed so that it always choose the best possible move.

[18] Tabu restrictions are also often used to prevent repetitions as well as reversals.

[19] In many applications, T is set to 7. In this case, once a move is added to the tabu list, it remains tabu for 7 iterations.

some attributes might contribute more strongly to a tabu restriction than others and, hence they should be given a briefer tabu tenure in order to avoid making the restrictions too severe [GLOV89a].

This "attribute approach" is not without problems. An attribute is not specific to one single solution. Therefore, once an attribute is pushed into the tabu list, it prevents the algorithm from visiting a number of solutions rather than masking one single point. Some of these solutions have never been visited before and are therefore perfectly valid; the algorithm should be able to visit these points. In this sense, it can be said that the tabu restrictions over-constrain the search. These non-desirable effects of the tabu restrictions are counter-balanced by the aspiration criteria.

The aspiration criteria.

The aspiration criteria is used to override the tabu restrictions. As these restrictions over-constrain the search, it is indeed necessary for the algorithm to have a mechanism which, on specific occasions, relaxes these constraints and allows a tabu move to take place. For instance, an algorithm may decide that if a tabu move results in a solution better than any solution visited so far, the tabu restrictions should be relaxed and the tabu move should be accepted [DEWE89]. In [LAGU91], the authors have observed that, in a run of 1000 iterations, an average of 7 tabu moves are executed as a result of meeting the aspiration criteria.

The medium-term memory module.

This module is used within TS *"to achieve regional intensification"* [GLOV89a]. In other words, this module tries to detect features such as values received by particular variables which are common to above-average solutions. In order to achieve this, the best solutions found by the short-term memory module are recorded and

compared. For instance, in a traveling salesman problem, the algorithm might then notice that some edges are often part of good solutions. Once this learning stage is completed, the algorithm uses this new knowledge to focus the search and tries to generate solutions that exhibit these features. This can be achieved by restricting the set of possible moves during the *period of regional search intensification* [GLOV89a].

The long-term memory module.

This module is used within TS *"to achieve global diversification"* [GLOV89a]. While the previous module tries to intensify the search in some promising regions of the search space, this module tries to direct the search towards regions which have not been explored yet; this technique gives a global view of the search space to the algorithm.

In this sense, TS is similar to GAs. Indeed, with their pools of solutions, GAs also have this global view. However, while GAs give a random sampling of the search space, TS (more precisely the long-term memory) tries to generate solutions which are definitely different from the past solutions. Glover describes the objective of this long-term memory as follows:

> "The objective is to create evaluation criteria that can be used by a heuristic search process which is specifically designed to produce a new starting point, thus generating such a point by purposeful instead of random means. These evaluation criteria penalize the features that long-term memory finds to be prevalent in previous executions of the search process." [GLOV89a].

3.4.4 Applications of TS.

Compared with GA and SA, introduced in 1975 and 1983 respectively, TS is still in its infancy and there are few applications of TS available. The same can be said for detailed or comparative studies of TS. Therefore, this subsection only reviews a limited number of applications.

In [MALE89], the authors performed two sets of experiments. In the first one, SA and TS are compared over a number of travelling salesman problems (up to 100 cities) and TS consistently outperforms SA. In the second set of experiments, the authors implemented parallel versions of TS and SA; in both cases, the overall system consists of a number of agents and each agent is solving the entire problem. In parallel TS, each agent performs a tabu search with different parameters, i.e. different attributes and aspiration criteria. Periodically, the agents are stopped and then the best solution is communicated to all the agents and the agents re-start from this new initial solution. In parallel SA, all the agents have a different annealing schedule. The main observation in this set of experiments is that both parallel TS and SA achieve a super-linear speed-up when compared with the original SA and TS. Since then, a similar phenomenon was reported in [CLEA91] where it is claimed that this super-linear speed-up was due to the fact that the different agents use the extra information (gained from cooperation with the other agents) to guide and focus the search in the most promising regions of the search space.

Perry also implemented a parallel TS for the solution of a time-constrained scheduling problem [20] [PERR90]. The problem is the allocation of surface-to-air-missiles to counter a number of threats. TS was used mainly because it is an iterative approach and hence, a solution can be provided (if needed) at any time. An initial solution is provided very rapidly, then, TS is used to refine this original solution.

[20] In this case, the main point of using a parallel algorithm is to increase the number of feasible schedules that can be generated in a given period of time.

Unfortunately, the authors provides few details concerning its application.

In [ADEN92], the authors compared the performance of TS against the performances of some greedy heuristics over 480 flow shop problems of different sizes. TS dominates its opponents in most cases. In [LAGU91], TS is used to solve a single machine scheduling problem. This problem is a sequencing problem and the objective is to minimise the sum of the set-up costs and delay penalties of the schedule. In this paper, the authors compare two moves operator for TS: swap and insert. The different algorithms are compared over some (relatively) small problems: 20 jobs, 35 jobs problems where the optimal solution can be found in less than 30 seconds. This study shows that insert is better than swap. Finally, a hybrid TS allowing both swap and insert moves was implemented. A second set of experiments showed that this hybrid version performed better than any one of the two original algorithms.

3.5 Brief comparison of GA, SA and TS.

GA, SA and TS originate from three different metaphors. GA replicates the principles of the theory of Evolution, i.e. selection and inheritance. SA is drawn from an analogy between thermodynamics and combinatorial problems and TS simulates three different types of memory (short-, medium-, and long-term).

Table 3.4 presents the key-features and the driving forces of these three techniques. First, they all are iterative and do not construct their final solution piece by piece. Rather, they start their search with an initial solution (or an initial population for GA) and gradually, iteration after iteration, improve the quality of that solution. Both GA and SA are stochastic and TS is, in most implementations, deterministic.

SA is a neighbourhood search technique; it works with only one solution and at

	GA	SA	TS
Type	global stochastic iterative	neighbourhood stochastic iterative	neighbourhood (and possibly global) deterministic iterative
Solution-candidate(s)	pool	one	one
Driving forces	initial population, crossover, mutation, selection function.	equilibrium, cooling rate, initial temperature.	tabu restrictions, aspiration criteria, medium-term memory, long-term memory.

Table 3.4: GA, SA, and TS: A comparison.

each iteration, this solution is only slightly altered. On the other hand, GA works with a pool of solutions and what happens to a particular chromosome is of little importance. This global view of the search space gives GA the opportunity to detect similarities among above-average solutions and then, GA can favour the propagation of these similarities during the search process.

When comparing SA and GA, it could be said that GA uses its population to *leap* from solution to solution in the search space whereas SA only goes from one solution to its neighbour. Our results in Chapter 6 show that these two algorithms have totally different computational behaviour.

Primarily, TS is a neighbourhood search technique; like SA, it uses a move operator to jump from neighbour to neighbour. However, the long-term memory module [21] provides TS with a global view of the search space. In contrast to GA, this *global view* is not provided by a random sampling of the search space; rather TS tries to generate (using a deterministic heuristic) new starting points which are radically different from past ones.

In conclusion, GA, SA, and TS are three AI search techniques which can be used

[21] Observe that this module is absent from most TS implementations.

to solve large combinatorial problems. They use their intelligence to explore large search spaces and find optimal or, more realistically, near-optimal solutions.

Chapter 4

Preliminary Studies.

This chapter describes two preliminary studies. First, a GA using an indirect representation is used to solve a simple scheduling problem. Then, a similar indirect GA is applied to a particular instance of the VRP. These two studies highlight the limitations of indirect approaches and advocate the adoption of a direct representation.

4.1 A simple prototype.

This prototype was used initially as an introduction to GAs. The objective of the GA was to find a schedule minimising the total tardiness [1] of a job set.

operations	A	B	C	D	E	F	G	H	I	J
duration	10	10	10	10	40	20	10	40	25	25
domain	20-50	20-40	30-40	20-99	50-200	80-200	100-200	0-200	0-150	0-150

Table 4.1: Description of a job set.

[1] Tardiness is the amount of time between the moment when a task should have been completed and the moment when the task is actually completed.

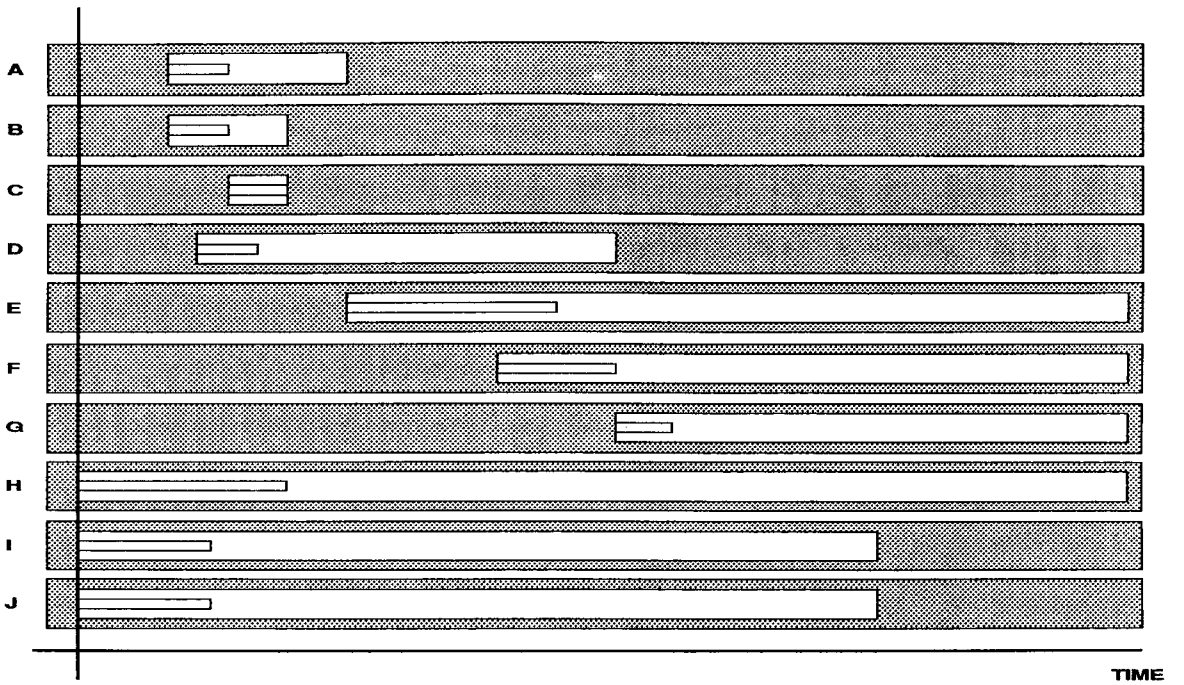

Figure 4.1: Current domains and durations of the operations.

Table 4.1 details the operations which make up the job set. The domain of an operation represents the set of possible values for the start-time of the operation, i.e. the time-window when the operation can be started. The different operations form a job set which must be performed on a single resource and this resource can only handle one operation at a time.

Figure 4.1 displays a Gantt chart which gives a graphical representation of this problem. The domains of the different operations are represented by a large horizontal white box. Within this large white box, a smaller box represents the duration of the operation.

The GA uses a steady-state model (described in section 3.2.2) and the representation employed for the chromosome is as follows:

$$o_1, o_2, ..., o_N$$

This is a simple order-based representation; o_i refers to the ith operation to be

instantiated. Each chromosome of the initial population is created randomly and represents one possible ordering of the job set.

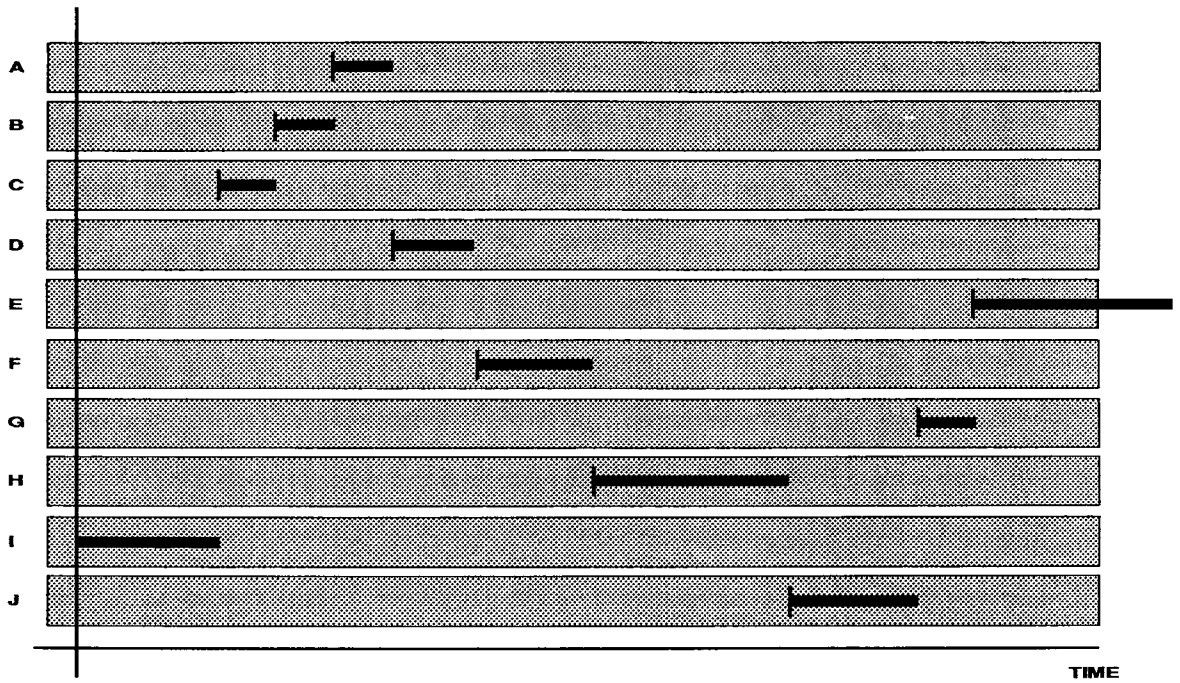

Figure 4.2: One optimal solution found by the genetic algorithm.

A schedule-builder uses the chromosome to generate a schedule; the order imposed by the chromosome is strictly followed and the schedule-builder assigns the earliest possible start-time to each operation. Each start-time has to be compatible with the previous decisions. If a start time cannot be found in the domain of the operation, the schedule-builder relaxes the problem and allocates a late start-time which is set to be as early as possible.

The different chromosomes are ranked [2] with respect to the total tardiness of their schedules and the probability of a chromosome being selected for reproduction is directly proportional to its rank in the population. The search is pursued until either an optimal solution, i.e. a solution with no tardiness, has been found or the maximum number of iterations has been reached.

[2]The best chromosome is the one which has the smallest tardiness.

The following table presents one optimal solution found by the GA with a population of 20 chromosomes and 500 iterations:

operations	A	B	C	D	E	F	G	H	I	J
start-time	50	40	30	60	175	80	165	100	0	140

Table 4.2: Optimal solution found by the genetic algorithm.

This optimal solution is also displayed in Figure 4.2; the solid black bars show the scheduling decisions, the length of the bar corresponds to the duration of the operation and its position represents the start-time of the operation.

During this project, an object-oriented approach has been used with Sun Common Lisp 4.0, CLOS (Common Lisp Object System), SPE (Symbolic Programming Environment 1.2) and Lispview 1.0 running on a SPARCstation IPC under Unix. Figures 4.3 and 4.4 show object representations of both a GA and a chromosome.

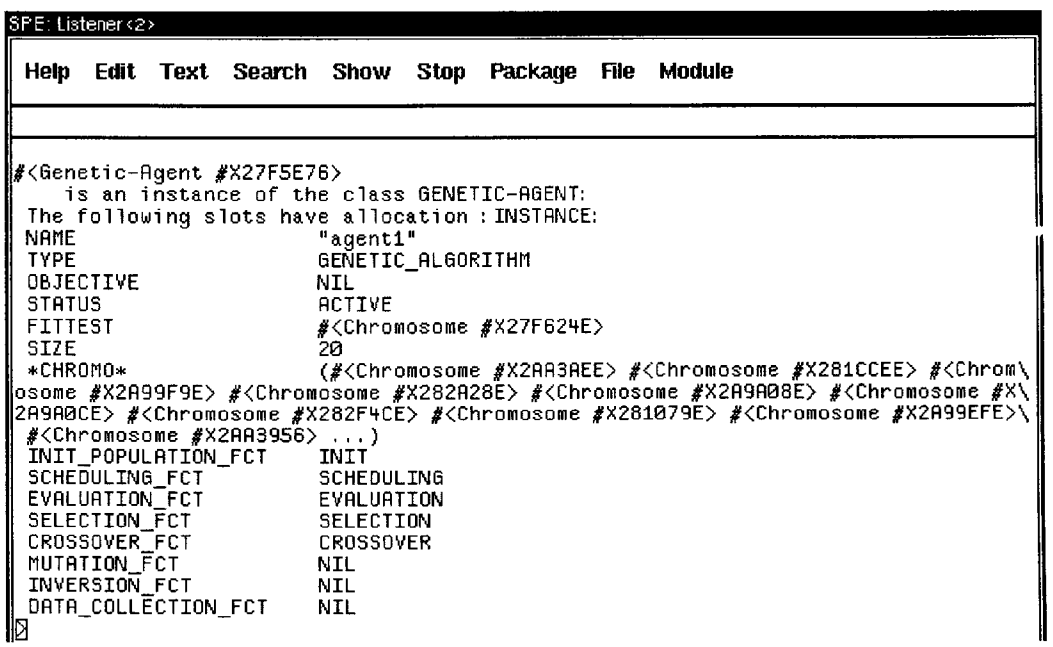

Figure 4.3: Object representation of a genetic algorithm.

Each instance of the class **genetic-agent** (see Figure 4.3) has the following slots:

name: unique name given by the user, when the GA is created, for manipulation purposes.

type: type of the object, i.e. genetic algorithm.

objective: documentation string describing the objective of the GA, e.g. *minimise total tardiness.*

status: status of the process associated with the object (i.e. init, active, inactive or terminated).

fittest: fittest element of the population.

size: size of the population, i.e. number of chromosomes in the population.

***chromo*:** list of chromosomes present in the population.

init_population_fct: function called by the process to create the population.

scheduling_fct: function called by the process to translate, i.e. map, order-based chromosomes into schedules.

evaluation_fct: function called by the process to evaluate the schedules.

selection_fct: function called by the process to select two parents from the population.

crossover_fct: function called by the process to perform the crossover operation.

mutation_fct: function called by the process to perform the mutation operation.

inversion_fct: function called by the process to perform the inversion operation.

data_collection_fct: function called by the process to collect data during the search.

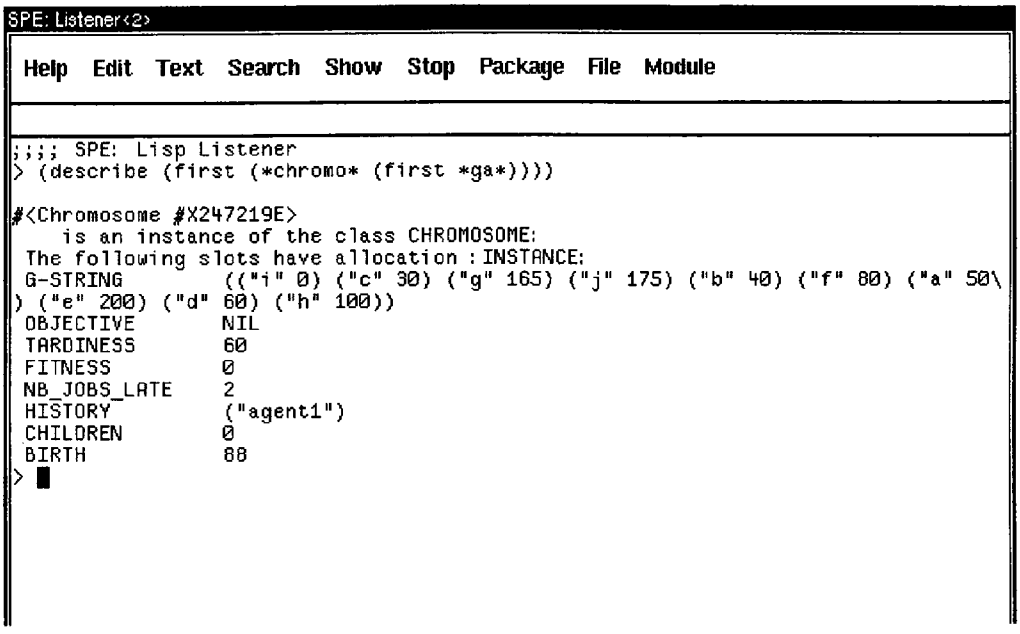

Figure 4.4: Object representation of a chromosome.

Each instance of the class chromosome (see Figure 4.4) has the following slots:

g-string: An order-based representation of the problem. Each couple of this string represents an operation and its allocated start-time.

tardiness: total tardiness of the jobs set.

fitness: fitness of the chromosome.

nb_jobs_late: number of operations that the scheduling function has not scheduled properly, i.e. number of operations with a late start time.

children: number of children of the chromosome.

birth: date of birth of the chromosome, i.e. number of cycles between the creation of the population (i.e. date 0) and the birth of the chromosome.

4.2 A particular VRP.

In this project, a GA is used to find near optimal solutions to one particular instance of the VRP. This VRP has the following parameters:

1. a workforce of 118 engineers operating from 11 bases,
2. a set of 250 jobs geographically distributed,
3. the objective of performing all the jobs while minimising the total distance travelled. The following cost function is used:

$$Cost = Travel + \sum_{i \in Conflicts} (Duration_i + 60)$$

where the first term represents the total distance travelled by the different engineers and the second term represents a penalty related to the conflicting jobs; for each job which has not been assigned to an engineer, the cost is increased by the duration of the job and a penalty of 60 points.

This initial problem has been further complicated by a number of side-constraints:

- Technological constraints (I): each job can only be performed by a sub-set of the workforce. On average, an engineer is only qualified to perform 57 jobs out of the 250 possible jobs. A significant number of jobs can only be performed by a few engineer. This may create some *bottleneck* situations.
- Technological constraints (II): the duration of a job depends on the engineer allocated to perform that job. The difference can be as large as 60%, say between an expert and an apprentice.
- Time windows (i.e, temporal constraints): a job can only be performed during a certain period of time. There are 3 classes of time-windows: all day, morning and afternoon.

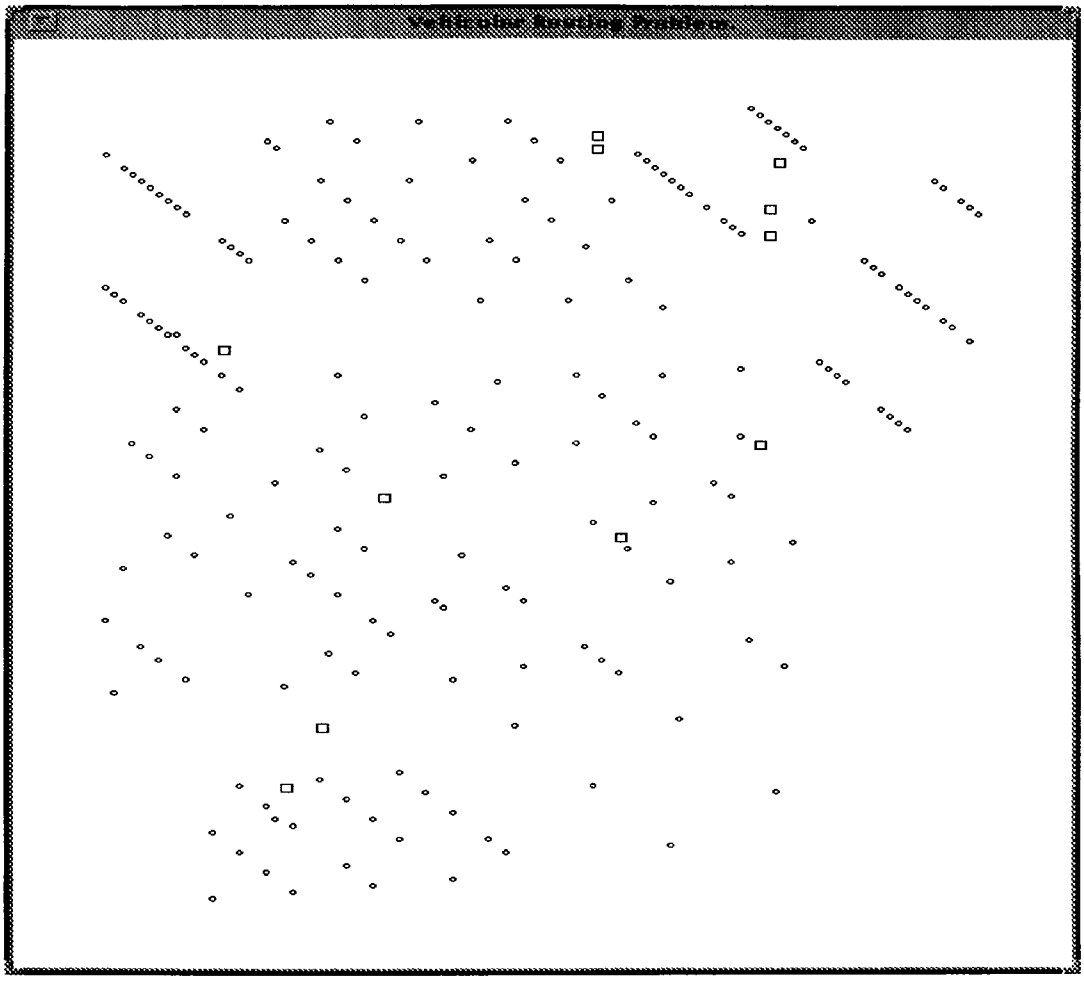

Figure 4.5: A particular VRP.

Figure 4.5 gives a graphical representation of this problem. The squares represent the 11 bases, where the 118 engineers start and finish their tours, and the circles represents the 250 jobs which have to be performed. All these elements are located within an area of 10 miles by 10 miles. Figure 3.1 in Section 3.1 displays a partial solution to this problem.

The GA works with an indirect representation of the problem; hence, it is used in association with a schedule-builder. As seen in Section 3.2.5 and in the previous prototype, the schedule-builder uses the chromosome as a decision sequence; it follows the order of the chromosome and makes a decision for every element of the chromosome. In fact, two indirect representations have been investigated in this

work:

1. The GA has a job-centred perspective. Each element of a chromosome represents a different job. The schedule-builder follows the ordered list and attempts to attach each job to an engineer. This strategy is similar to the operation-centred perspective adopted by ISIS [FOX84].

2. The GA has an engineer-centred perspective: each element of a chromosome represents a different engineer. The schedule-builder attempts to build a tour for each engineer successively. This strategy is similar to the resource-centred perspective adopted by OPIS [SMIT90].

   The schedule-builder works as follows. First, it assumes that all the jobs are conflicting (as soon as a job is allocated to an engineer, it is removed from the set of conflicting jobs). The algorithm attempts to attach the current engineer to all the jobs he/she can do and which are still in the conflict list, i.e. jobs which have not been allocated yet. Then, the algorithm moves to the next engineer in the chromosome.

The two strategies are illustrated in Figure 4.6. Two GAs have been implemented: one with a job-centred perspective and one with an engineer-centred perspective. Both GAs have the following set-up:

- population: 100 chromosomes.
- crossover: The PMX crossover operator (described in Section 3.2.5).
- mutation: The mutation operator works with a single chromosome; two elements of the chromosome swap position. This operation is performed after the crossover operation, on 4% of the children.
- inversion: The inversion operator also works with a single chromosome; it inverts the order of the elements between two randomly-chosen points of the

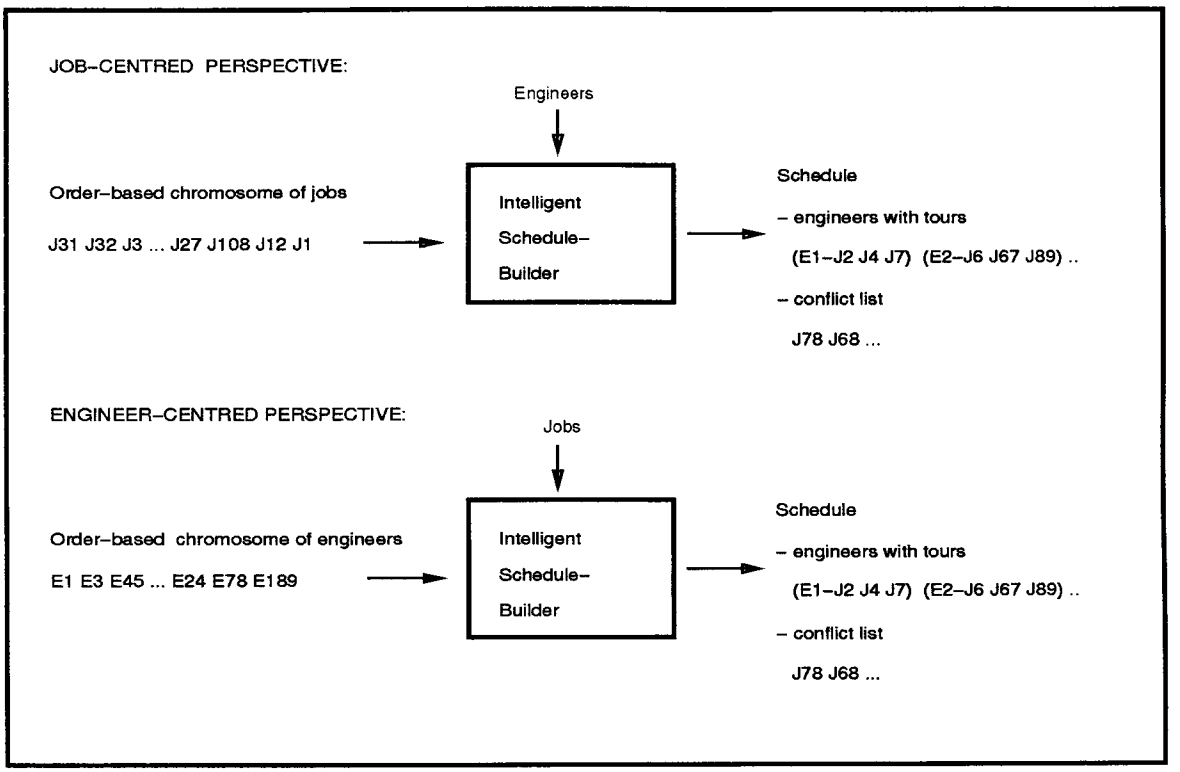

Figure 4.6: Job-centred and engineer-centred perspectives.

chromosome. This operation is performed after the crossover operation, on 4% of the children.

- Termination criteria: The GA is stopped after 100 iterations which is around 20 minutes on a SPARCstation IPC with LISP.
- Model: The two GAs use a steady-state model and the chromosomes are selected according to their rank in the population.
- Due to the stochastic nature of GAs, each algorithm is run 20 times.

Experiments were performed (1) to examine the quality of the schedules produced by the two different schedule-builders, i.e. job- and engineer-centred, and (2) to examine the performances of the two GAs.

Perspective	μ	min	max	σ
JOB	24387	22770	25954	523
ENG	23168	22097	25191	419

Table 4.3: Performances of the two scheduling functions.

For each schedule-builder, a set of experiments has been carried out to check the quality of the schedules produced. In each case, 1000 random strings have been generated and processed through the schedule-builder. In the case of the job-centred function, each string was a random permutation of the list of jobs. For the engineer-centred function, each string was a random permutation of the list of engineers.

Table 4.3 presents the performances of the two schedule-builders. The column μ gives the average cost achieved by the function (out of 1000 tries), min gives the minimum, max gives the maximum, and σ gives the standard deviation.

On average, the engineer-centred function produces better solutions than the job-centred function. The main reason is that the engineer-centred function can handle *bottleneck* engineers in a more efficient manner and, hence, can schedule more jobs than the job-centred function. Observe that the two standard deviations are relatively small compared to the average values. This suggest that the schedules are quite similar to each other. This might limit the performances of both GAs, which might not have enough information (more precisely, enough variety in their respective pool of solutions) to guide their search through the solution space in an efficient manner.

The two curves, displayed in Figure 4.7, show how the GAs performs on this particular problem. As expected, the engineer-centred GA dominates its opponent from start to finish. The two curves are concave and as the search goes on, the average additional improvement found per iteration diminishes. At the end of the search

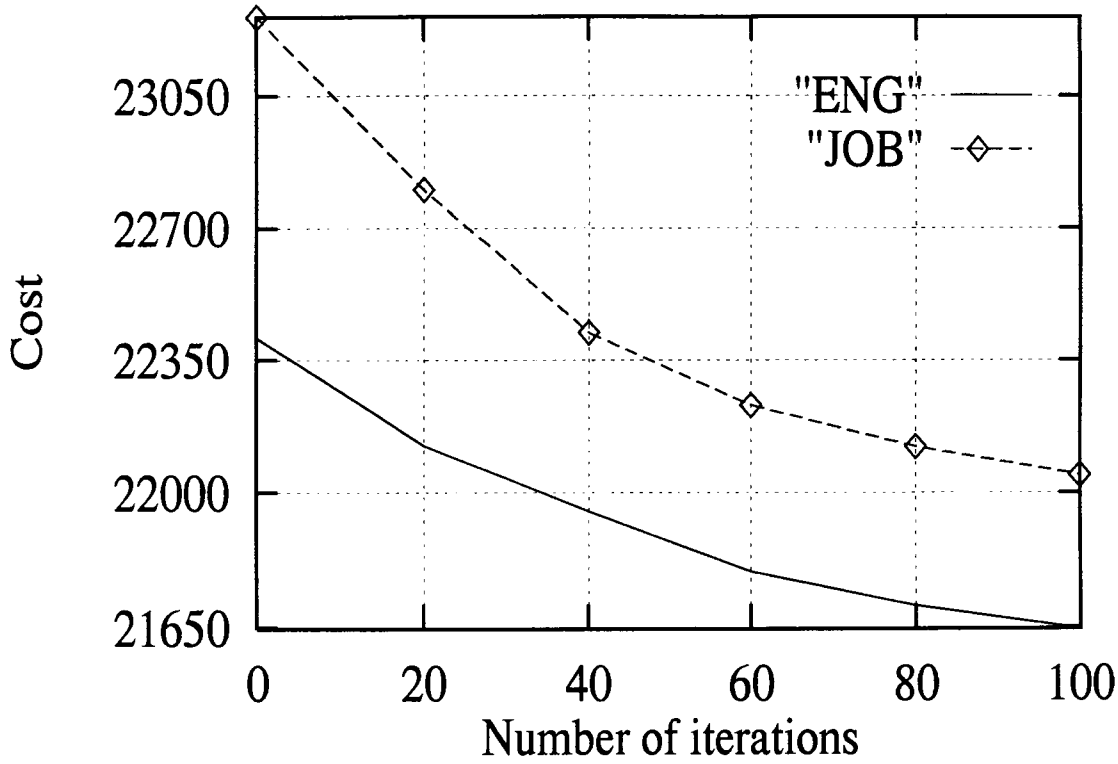

Figure 4.7: Evolution of performances with time.

(in this case, after 100 iterations), the improvements are marginal. This suggests that both GAs have converged towards a solution. The main reason for such an early convergence is the relatively small size of the populations.

4.3 Conclusion.

The two studies presented in this chapter were used (originally) as an introduction to GAs. In order to solve the two problems under study, a traditional and indirect approach was adopted. The GA, in both cases, was working in association with a schedule-builder.

In the first study, this association (GA and schedule-builder) was capable of reaching optimal solutions, i.e. solutions with no tardiness. This was facilitated by the

fact that the algorithm only had to explore a relatively small search space. In the second study, both GAs (with job-centred and with engineer-centred perspectives) were capable of improving on their initial solutions.

However, the indirect approach has some important drawbacks. First, this type of representation may not allow a total exploration of the search space, i.e. it may compromise *completeness*. This is because the GA works with an indirect view of the problem, e.g. list of jobs, and it may be the case that the optimal solution, or more generally, good quality solutions, cannot be produced by the schedule-builder from *any list of jobs*.

Moreover, despite the fact that indirect GAs have been successfully applied to a wide range of sequencing problems such as the traveling salesman problem, there is no guarantee that this approach will be equally successful when applied to problems that are not pure sequencing problems.

The last point can be explained as follows. Indirect GAs tend to preserve the relative order of the elements in the chromosome during a crossover operation; the main assumption is that the solution-builder will generate good quality solutions, if provided with well-ordered chromosomes. This indirect strategy is likely to be disruptive when applied to problems which are not (100%) sequencing problems. Clearly, indirect GAs might not guide their search properly. In the worst case, a schedule-builder might generate the same solution for any ordered list. This will be the case if the elements within the list are identical. For example, assume an indirect GA working with an engineer-based perspective where all the engineers are identical and are therefore inter-changeable. Any possible permutation of this ordered list will generate the same schedule (or more exactly, schedules with the same cost).

Finally, the schedule-builder generates new solutions entirely from scratch at each crossover. This new solution is then entirely evaluated from scratch. These two

operations are extremely expensive in terms of computational effort.

Again, the above limitations advocate the implementations of a direct approach similar to the ones reported previously by Bagchi et al. [BAGC91] and Bruns [BRUN93].

More generally, these two studies also suffer from a number of important limitations. First, the performances of the algorithms are not compared against the performance of random search; neither are they compared against the performances of other search techniques. Hence, it is not possible to know if the algorithm guides its search in an efficient manner or if the algorithm is just another form of random search; neither is it possible to know how the algorithm compares relatively to other search techniques.

Another important limitation of these two projects is the fact that only one problem is studied. Here, there is the danger of designing an algorithm that is very good at solving a particular problem but nothing else.

In conclusion, an algorithm should be tested against a wide range of problems, each with different characteristics. Its performance should also be compared against those of other techniques. It will then become possible to define the most suitable techniques and parameters for each class of problems. It will also be possible to see how robust a particular technique is, to determine how well it can handle different classes of problems. Finally, such an extensive study should provide enough information about the different techniques and about the problems to anticipate the behaviour of the algorithms and the quality of their answers.

The next two chapters report three sets of experiments which follow the above recommendations. In this study, direct GAs are applied to a wide range of vehicle routing problems. These problems have a substantial range of variation in each of three dimensions, i.e. work load, time constraints and specialisation constraints. Moreover, the performances of these direct GAs are compared against the

performances of other search techniques.

Chapter 5

Experimental Set-up.

This chapter is divided into four sections. In Section 5.1, the set of vehicle routing problems used for the experiments is described. Then, Section 5.2 presents the common model - or data structure - that has been used to represent the solutions of the different problems. Section 5.3 defines the objective function and Section 5.4 introduces a new class of GAs - especially designed for the solution of VRPs - which overcomes many limitations of traditional GA-based systems.

5.1 The set of problems.

Thirty six problems with diverse characteristics have been generated. In order to perform a study as wide ranging as possible, extremes classes of problems have been considered. In particular, this work has examined how three parameters: workload, time constraints (also called time windows) and specialisation constraints (also called technological constraints) can affect the performance of different search techniques.

A problem can be under-resourced, critically resourced or over-resourced, it can have loose or tight time constraints and loose or tight specialisation constraints. Thus, there are twelve different parameter setting and three problems are created at each parameter setting, hence, the thirty six problems. These twelve parameter settings and the associated thirty six problems can be positioned on different points of a cube (see Figure 5.1). On this cube, each point represents three problems and one parameter setting.

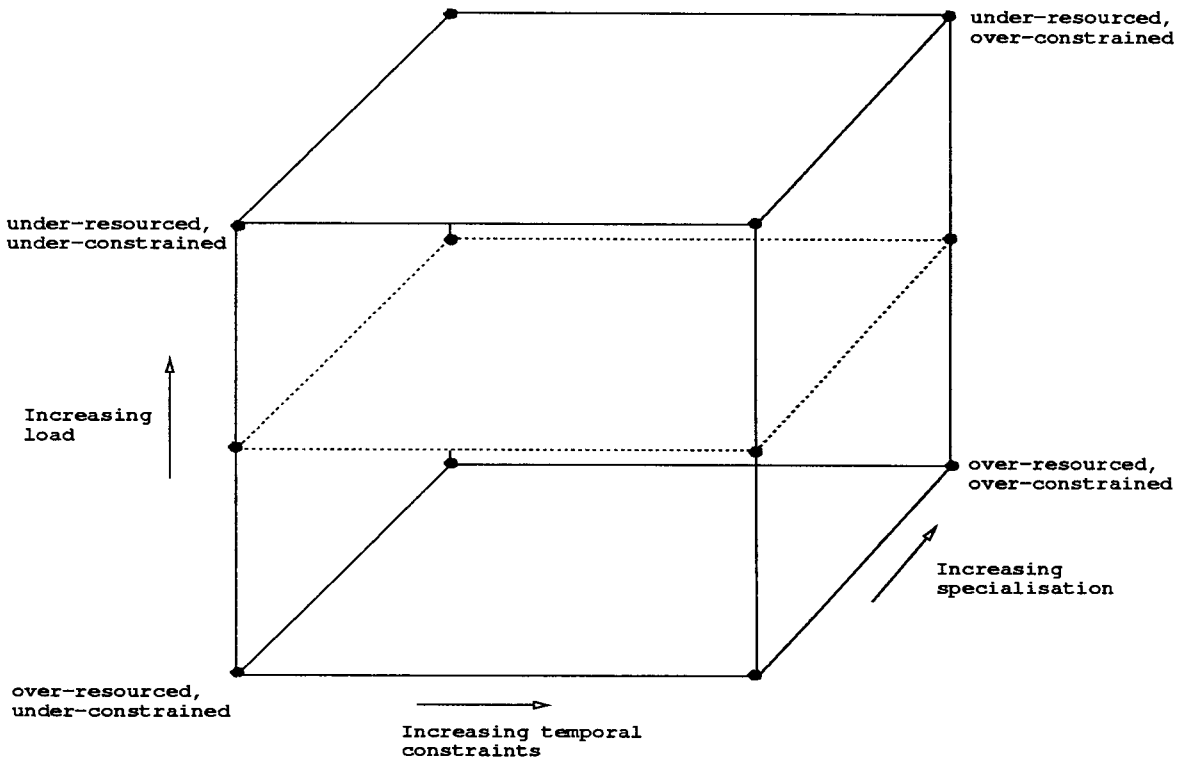

Figure 5.1: The 36 problems.

The 6 points on the left face of the cube represent problems with loose time windows. As we move down an edge (any edge) we move from under-resourced problems to over-resourced problems. Moving left to right we increase the tightness of the temporal constraints, and all problems on the right hand face have the tightest temporal constraints. Moving into the page we increase the tightness of the specialisation constraints.

A different data set [1], i.e. engineers and jobs, is generated randomly for each problem, and this data set respects the following rules:

- All the jobs are located within a 5 kilometres radius of a central location. Hence, all the 36 problems are urban.
- Several jobs can shared the same location.
- All problems have 200 jobs, the durations of these jobs are drawn randomly from the distribution {15,30,45,60,75,90,105,120 minutes} and each duration is equally likely.
- 10% of the jobs are compulsory. If one of these high priority jobs is not allocated, then the search algorithm must create some space in the schedule, for example, by removing one of the low priority jobs.
- The level of time windows (T) is either loose ($T = 0$) or tight ($T = 2$). If $T = 0$, all jobs can be done at any time during the working hours. If $T = 2$, 45% of the jobs can only be done in the morning, 45% can only be done in the afternoon, and 10% can be done at any time - morning and afternoon.
- Under-resourced problems have 30 engineers, critically resourced have 40 and over-resourced have 50.
- The level of specialisation constraints (S) is either loose (S=0) or tight ($S = 3$). If $S = 0$, all jobs can be done by any engineer, and if $S = 3$, 80% of the jobs can be done by 20% of the engineers, and 20% of the jobs can be done by 80% of the engineers.
- There is only one base station. All the engineers start their tours from this base station after 9am and come back to this base station for 5pm.

[1]The problem generator which was used to create these problems is presented in more detail in Annexe I.

Engineers travel at 12mph. Manhattan distances are assumed and engineers travel up and down, left and right, but not diagonally. Working hours are 9am to 5pm (or 540 to 1020 measured in minutes from midnight). Morning is from 9am to noon (or 540 to 720) and afternoon is from noon to 5pm (or 720 to 1020). No overtime is allowed, engineers must be back at the base station for 5pm.

From now on, the following convention will be used to refer to a problem or to a set of problems:

problem E-T-S-N

where E indicates the number of engineers, T and S indicate the tightness of the time windows and of the specialisation constraints, and N is a number used to distinguish between the three problems generated at each point of the cube. For instance, the expression "*problems 30-*-2-**" refers to the set of problems with 30 engineers and tight specialisation constraints.

5.2 An active constrained-based model.

All the search techniques use a common representation of the VRP. A solution is represented as a list of engineers, and each engineer has his own ordered list of jobs (his tour). For an n engineer problem, there is always the $n + 1th$ engineer (the virtual engineer) that is able to do all work. Therefore, if a job has not yet been allocated to any of the n engineers it will reside in the tour of the $n + 1th$ engineer. This tour is frequently referred to as the "conflict list" or as the set of "jobs not done".

A constraint-based approach [2] is taken when representing an engineer's tour. A tour is represented as a sequence of jobs, such that it is possible to perform each job

[2]This mechanism is required so that the search process does not take an over-constrained

within its time window, travel between jobs in the given sequence, and return to the base before the end of the day (i.e. a legal tour using a least commitment strategy). When a job is inserted into a tour, constraint propagation takes place, updating (restricting) the time windows of the jobs within that tour. Symmetrically, when a job is removed from a tour, constraints are retracted, and the time windows may be relaxed. Therefore, central to the representation of the VRP is a constraint maintenance system that gives an active representation of tours. This constraint maintenance system was implemented by Craig Brind.

5.3 The cost function.

The principal objective is to perform all the jobs and a second objective is to minimise the total distance travelled by the workforce. The following cost function is used to attribute a cost for each solution:

$$cost(s) = C.D(s) + T(s)$$

where C is a constant, $D(s)$ is the sum of the jobs not done in minutes, and $T(s)$ is the sum of the travels in minutes of the engineers. The travel time for an engineer assumes *Manhattan distance* [3] covered at a given speed (in this case 12mph), and is measured in minutes.

In some early tests, a small value had been chosen for C ($C = 3$). However, in some instances, the different algorithms were trading work for travel, i.e. algorithms could get a lower cost by doing less work and, hence, much less travel. This situation was undesirable. The objective of performing all the jobs is indeed more important

view of the problem (resulting in lost opportunities) or an under-constrained view of the problem (resulting in infeasible solutions).

[3] Given two locations $< x_1, y_1 >$ and $< x_2, y_2 >$, the Manhattan distance is $|x_1 - x_2| + |y_1 - y_2|$.

than the reduction of travel, and the cost function has to reflect this obligation.

By choosing a large enough value for the coefficient C we hope that it is not possible to trade travel for work done. Put another way, given any two solutions s' and s'', where $cost(s') < cost(s'')$, $D(s') \leq D(s'')$. Therefore, we can make an improvement to s by increasing the amount of work done, i.e. reduce the amount of work *not* done, or by reducing the amount of travel. It should be impossible to reduce the cost of s by reducing the amount of work done in s (increasing the amount of work *not* done in s).

All the experiments were run using a coefficient $C = 600$. Hence, the cost of a solution is 600 times the durations of the jobs not done plus the different travels and, problems with large cost solutions are typically under-resourced, i.e. work is left undone.

5.4 Direct GAs.

The GAs presented in this section are designed with the principal objective of overcoming the many limitations of traditional GA-based systems. This research follows the initial works of Bagchi et al. and Bruns reviewed in Section 3.2.5. These GAs work directly [4] on the solution-candidates, i.e. the schedules are the chromosomes. There is no longer any schedule-builder. More precisely, the scheduling function is performed directly by the crossover operator. Each element (or locus) of a chromosome represents one engineer, and each one of these loci can be assigned different tours (or alleles). During the reproduction process, parts of two schedules are mixed together to produce one new schedule. Direct crossovers can be split up into two broad categories:

[4] From now on referred to as the "direct" GAs.

- Direct crossovers using a random-based inheritance procedure, where a chromosome inherits random parts of both parents.

- Direct crossovers using a knowledge-based inheritance procedure, where a chromosome inherits the best parts (with respect to one or several given parameter(s)) of its parents. A knowledge-based crossover gives more intelligence to the GA; the operator identifies the best parts, i.e. tours, of a schedule and favours their inheritance during the crossover operation.

Different knowledge-based crossovers can be envisaged, e.g. a crossover may favour the inheritance of tours with a large number of jobs, tours representing a large amount of work, or tours with a low percentage of travel. For instance, when solving under-resourced problems, a GA may attempt to preserve tours with large amounts of work being done. These tours are likely to be well "packed" and, hence, are likely to represent good building blocks for the creation of good-quality solutions.

However, it is expected that these crossovers may not be as robust as random-based crossovers. For instance, in the case of over-resourced problems, the previous crossover, trying to preserve tours with large amounts of work, may not be desirable. Here, all jobs will be performed as the problem is over-resourced and, hence, the primary objective will become the reduction of travel. This crossover will misguide the search. In this case, a random-based crossover [5] will be more adaptable or rather, less deterministic and will reach a better solution.

In this study, a number of direct crossovers have been implemented and tested. They are now presented in some detail:

- Crossover #1 and #2 use the same overall algorithm. The principal difference is that Crossover #1 uses a random-based inheritance procedure and

[5] entirely driven by random decisions.

Crossover #2 uses a knowledge-based inheritance procedure.

- Crossover #3 and #4 attempt to overcome some limitations shown by both Crossover #1 and #2. They are both based on the same overall algorithm; Crossover #3 uses a random-based inheritance procedure and Crossover #4 uses a knowledge-based inheritance procedure.

Observe that these crossovers use the tours of two chromosomes, i.e. the *parents*, to produce one chromosome, i.e. the *child.*

5.4.1 Crossover #1

Initially, two parents have been selected for reproduction and the *child* is empty, i.e. all the jobs are in the child's conflict list. This crossover is a three stage process:

1. **Inherit from** $parent_1$: the child inherits half its tours from $parent_1$. For example, if the problem is under-resourced and there are 30 engineers, 15 tours are copied from $parent_1$ to the *child.* All the jobs present in these tours are then removed from the child's conflict list.

2. **Inherit from** $parent_2$: for each job still in the child's conflict list, the crossover tries to allocate the job to the same engineer as in $parent_2$. For example, if job J_6 is done by engineer E_3 in $parent_2$, then the crossover tries to allocate job J_6 to engineer E_3 in the *child.* All jobs which are successfully allocated are removed from the child's conflict list.

3. **Repair:** for each job still in the child's conflict list, the crossover tries to allocate the job to any engineer. Again, all jobs which are successfully allocated are removed from the child's conflict list.

```
1   FUNCTION Crossover1(parent1, parent2)
2   BEGIN
3   child = new-chromosome();
4   child = inherit-half(parent1);
5   FOR job IN conflict-list(child)
6   DO BEGIN
7       decision = nil;
8       eng = who-is-doing(job,parent2);
9       IF eng != CONFLICT-LIST
10      THEN decision = try-to-allocate(job,eng,child);
11      IF decision = true
12      THEN remove-from-conflict-list(job,child);
13  END
14  FOR job IN conflict-list(child)
15  DO BEGIN
16      decision = nil;
17      FOR all engineers While decision = nil
18      DO BEGIN
19          decision = try-to-allocate(job,eng,child);
20          IF decision = true
21          THEN remove-from-conflict-list(job,child);
22  RETURN child;
23  END;
```

Figure 5.2: Pseudo-code for Crossover #1.

The pseudo-code for Crossover#1 is given in Figure 5.2 and Figure 5.3 provides a graphical description of the actions taken during a Crossover#1 operation. In this example, there are 4 engineers, E_1 to E_4, and 20 jobs to allocate, J_1 to J_{20}.

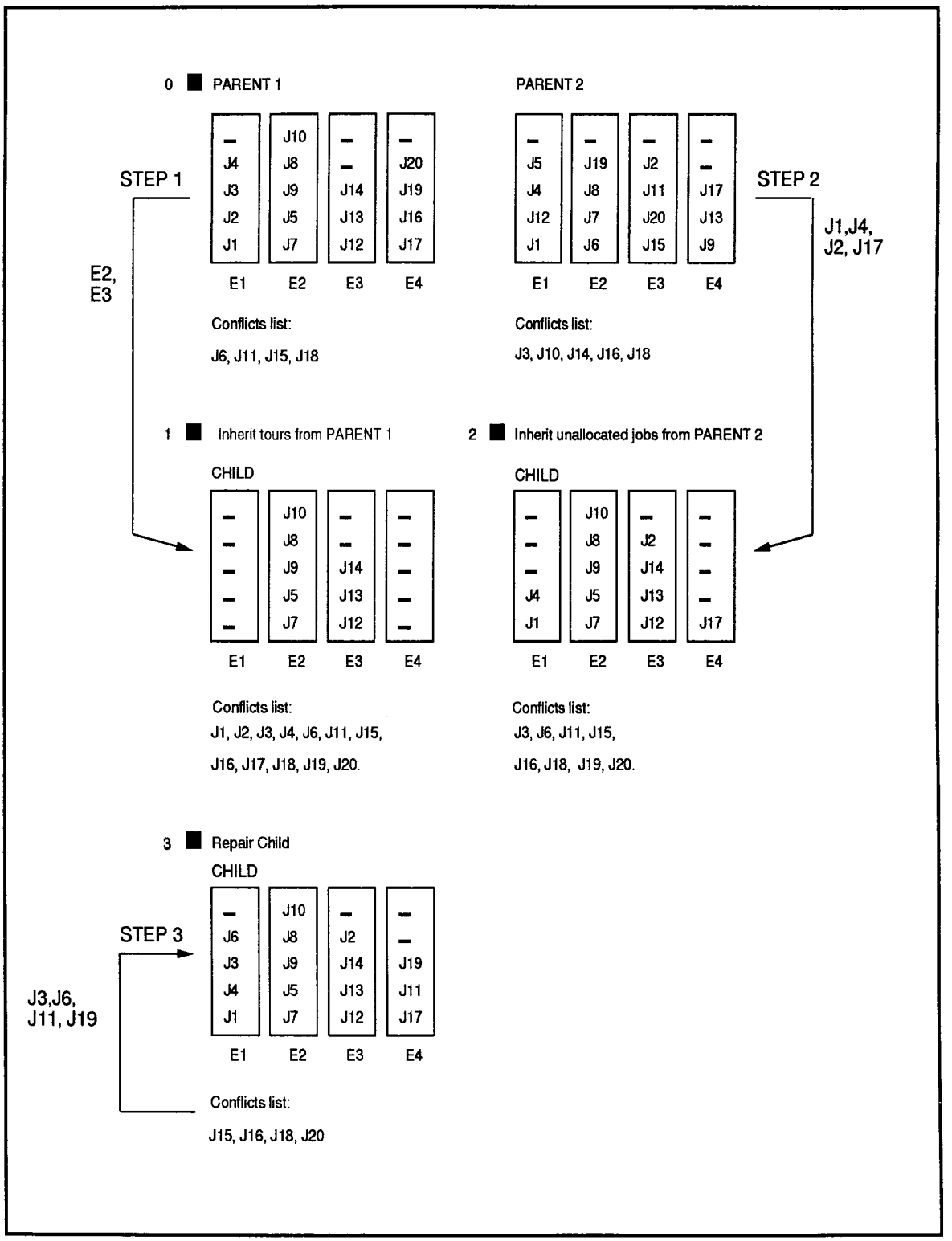

Figure 5.3: Crossover #1, step by step.

In the first stage, the *child* inherits half its tours from $parent_1$. There are four engineers so two tours are copied, tours of engineers E_2 and E_3 have been selected randomly; they are now copied from $parent_1$ into the *child* (step 1). At this point, twelve jobs are still in the child's conflict list. For each of these jobs, the crossover tries to allocate the job to the same engineer as in $parent_2$. In this example, it has been possible to allocate jobs J_1, J_2, J_4, and J_{17} to the same engineer as in $parent_2$ (step 2). There are still eight jobs remaining in the child's conflict list. The crossover now tries to allocate them to any possible engineer. Jobs J_3, J_6, J_{11}, and J_{19} are inserted during this operation (step 3).

5.4.2 Crossover #2.

This crossover follows the same sequence of steps as Crossover #1, but uses a knowledge-based inheritance procedure. That is, the *child* still inherits half the tours of $parent_1$. However, the inheritance is no longer random-based. It is now knowledge-based and the *child* inherits the best tours of $parent_1$, i.e. the child inherits the best half of $parent_1$. Again, the definition of *best tour* might be related to the number of jobs in a tour, the sum of the durations of the jobs in a tour, or some measure of *profit* (whatever that measure may be). The rest of the algorithm, i.e. steps 2, 3, and 4, is identical to Crossover #1.

Separate tests - done in the early stages of the implementation - showed that the above crossovers (#1 and #2) are extremely disruptive and a GA equipped with either one of these crossovers performs only marginally better than random search.

5.4.3 Crossover #3.

Crossover #3 follows a three stage process:

1. **Inheritance from** $parent_1$: the entire chromosome, i.e. tours and conflict list, is copied from $parent_1$ to the *child*. At this point, the *child* is a clone of $parent_1$.

2. **Inheritance from** $parent_2$:

   (a) The crossover now decides which tours will be copied from $parent_2$ to the *child*. 50% of the tours in $parent_2$ must be inherited by the child. For instance, if the problem has four engineers, the crossover may decide randomly that the tours of E_2 and E_3 will be copied.

   (b) Before the crossover can actually copy these tours from $parent_2$ to the *child*, it must work on the child so that future duplications and omissions of jobs in the *child* can be avoided:

      i. The selected tours (e.g E_2 and E_3) are deleted from the child's schedule and their jobs are pushed into the child's conflict list.

      ii. Jobs allocated to the selected engineers in $parent_2$ are identified. All these jobs are then deleted from the *child* both in the child's tours and in the child's conflict list.

      iii. Once this prerequisite work is done, the selected tours are copied from $parent_2$ to the *child*.

3. **Repair** *child*: for each job present in the conflict list, the algorithm attempts to allocate the job to any engineer. As usual, all jobs which are successfully allocated are removed from the child's conflict list.

The pseudo-code for Crossover #3 is given in Figure 5.4 and Figure 5.5 provides a graphical description of the actions taken during a Crossover #3 operation.

Figure 5.5 shows how Crossover #3 works. First, the entire schedule is copied from $Parent_1$ to *child* (step 1). Then, a random decision on which tours are to be copied from $Parent_2$ to *child* is taken. Here, tours 2 and 3 will be inherited from $Parent_2$ (step 2). At this stage, the algorithm removes all the jobs from the child's tours 2 and 3 and pushes them into the child's conflict list. Jobs J_5, J_7, J_8, J_9, J_{10}, J_{12}, J_{13}, and J_{14} are moved to the child's conflict list (step 3). All the jobs allocated to tours 2 and 3 in $Parent_2$ are deleted from the *child* (both tours and conflict list); jobs J_2, J_6, J_7, J_8, J_{11}, J_{15}, J_{19}, and J_{20} are deleted (step 4). *Child* inherits the selected tours from $Parent_2$; tours 2 and 3 are copied from $parent_2$ (step 5). The final action is to repair the child and attempt to insert unallocated jobs into the schedule; here, jobs J_5, J_9, J_{10}, J_{12} are inserted.

Crossover #4.

This crossover follows the same sequence of steps as Crossover #3, but, like Crossover #2, it uses a knowledge-based inheritance procedure and the child inherits the best half of $Parent_2$. The rest of the algorithm, i.e. steps 2, 3 and 4, is identical to Crossover #3.

The performance of these direct GAs are examined in the next chapter.

```
1   FUNCTION Crossover3(parent1, parent2)
2   BEGIN
3   list-of-tours = tours-selection();
4   FOR tour IN list-of-tours
5   DO BEGIN
6       remove-tour(tour,child);
7   END
8   FOR tour IN list-of-tours
9   DO BEGIN
10      list-of-jobs = push-jobs(tour,parent2)
11  END
12  FOR job IN list-of-jobs
13  DO BEGIN
14  delete-job(job,child)
15  END
16  FOR tour IN list-of-tours
17  DO BEGIN
18      copie-tour(tour,parent2,child)
19  END
20  FOR job IN conflict-list(child)
21  DO BEGIN
22      decision = nil;
23      FOR all engineers While decision = nil
24      DO BEGIN
25          decision = try-to-allocate(job,eng,child);
26          IF decision = true
27          THEN remove-from-conflict-list(job,child);
28  RETURN child;
29  END;
```

Figure 5.4: Pseudo-code for Crossover #3.

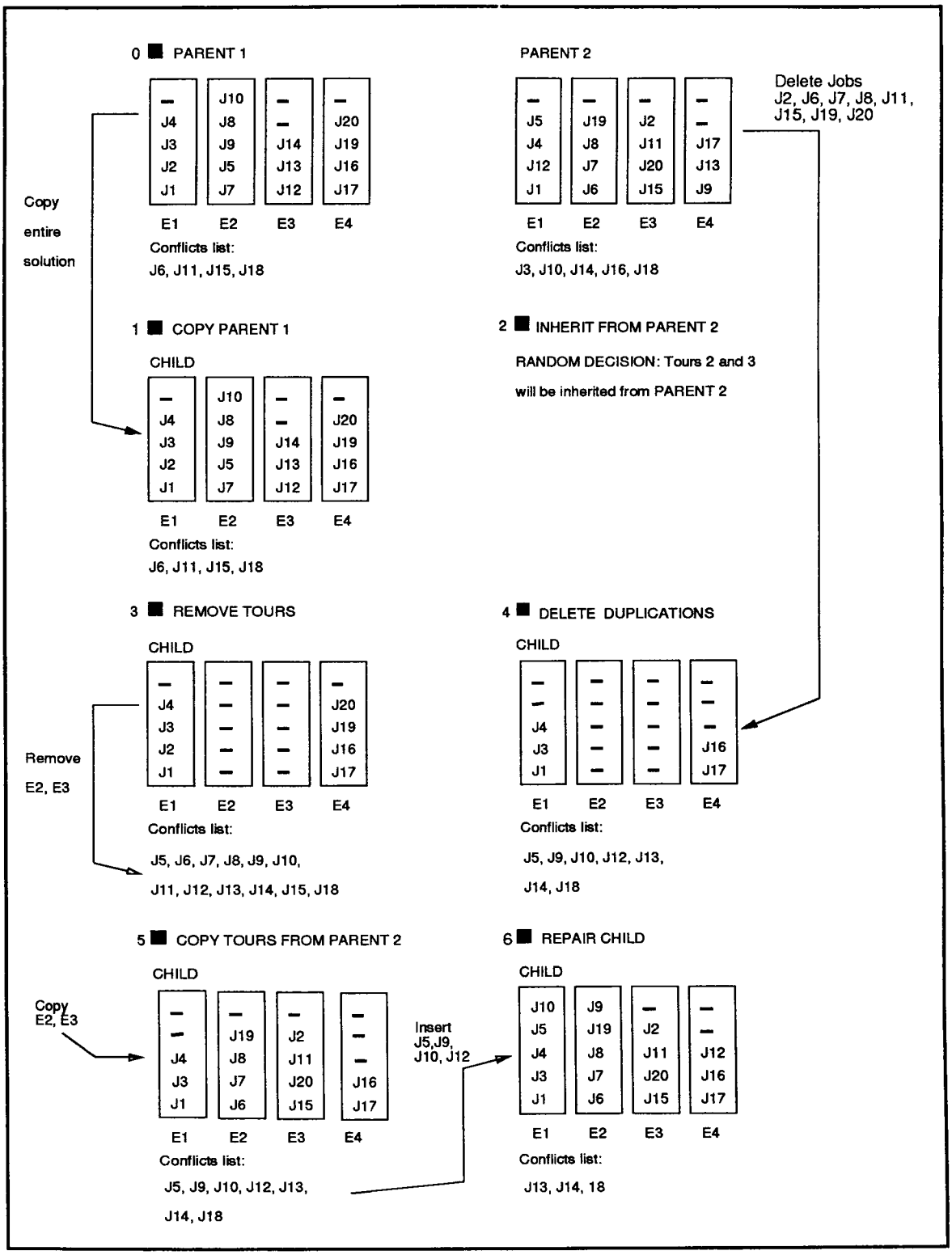

Figure 5.5: Crossover #3, step by step.

Chapter 6

Experiments and Discussions.

This chapter is divided into three sections; each section reports a separate set of experiments:

1. In Section 6.1, the direct GAs introduced in Chapter 5 are compared against an indirect GA, and random search. Two models - steady-state and generational - are implemented. A study is then performed to determine the effect of different population sizes on a given direct GA. Finally, this section reports on the performance of a direct GA associated with a local repair algorithm. Each technique is given ten minutes of CPU time (on a SPARCstation IPC) per problem.

2. In Section 6.2, the best direct GA is compared against random search, hill climbing, simulated annealing and tabu search. Again, each technique is given ten minutes of CPU time per problem.

3. Section 6.3 repeats the previous comparative study. However, in this case, each technique is given the equivalent [1] of four hours of CPU time per problem.

[1] In fact, these CPU intensive experiments were executed on a DEC Alpha 3000/300; each algorithm was given one hour of CPU time and the DEC Alpha was roughly four time faster than

The different algorithms were written in C and runs were performed on two SPARC-stations IPC and one DEC Alpha 3000/300 under the UNIX operating system.

6.1 Genetic Algorithms.

This section examines GAs in some detail and addresses a number of issues:

1. Which GA is the best among a set of possible models and crossovers. The behaviour of several GAs are studied over a set of problems and the conditions under which one GA is better than another are defined.

2. The effect of different population sizes on the performance of direct GAs.

3. A repair algorithm is associated with a direct GA and the behaviour of this new hybrid algorithm is studied.

During this study, each algorithm has been applied 5 times to each problem and the start value of each run was chosen randomly, so that an indication of the average behaviour can be obtained. Comparisons between algorithms are based on the average values resulting from these trials. CPU time is limited such that each algorithm is only allowed 10 minutes on each problem. Therefore, to investigate the performance of an algorithm on a single point of the cube takes about 150 minutes - there are 3 problems at each point, and the algorithm is applied 5 time to each problem, 10 minutes for each application. To apply a single algorithm to all 36 problems takes 30 hours and 12 algorithms are examined in this section. Therefore, it took approximately 360 hours on a SPARCstation IPC to perform the set of experiments reported in this section.

---

a SPARCstation IPC.

6.1.1 Comparison of different genetic algorithms.

During this study, a number of GAs have been implemented. Here, we examine the performances of these different algorithms and compare them over the entire range of problems. Four GAs are compared and RS is used as the reference algorithm:

1. The first GA is the only indirect GA of this comparative study; it uses a generational model and the partially mapped crossover [2] (PMX). The chromosome is an ordered list of engineers. The schedule-builder [3] takes the ordered list of engineers and allocates jobs to each engineer in turn. This first GA will be referred to as PMX.

2. The second GA is a direct GA; it uses a generational model and the random-based direct Crossover #3. This GA will be referred to as Direct3.

3. The third GA is also a direct GA; it uses the steady-state model and the random-based Crossover #3. This GA will be referred to as Direct3-s.

4. The fourth GA is again a direct GA; it uses a steady-state model and the intelligent direct Crossover #4. This crossover uses the amount of work done to sort the different tours of a solution and tries to preserve the best tours. This GA will be referred to as Direct4-s.

5. RS performs a random sampling of the search space and keeps a record of the best solution found during the search. RS enters a loop and at each iteration the algorithm creates a random solution via the schedule builder, and then evaluates that solution. The cost of the current solution S is then compared with the cost of the best solution found so far (S_{best}). If $cost(S) <$

[2] This order-based crossover attempts to preserve the absolute positions of elements for one part of a child and the relative order for the other part. This crossover is described more fully in Subsection 3.2.5

[3] This schedule-builder is used by RS to generate all its solutions and also by all the direct GAs to create their initial populations.

$cost(S_{best})$ then S_{best} is replaced by S, otherwise S_{best} remains unchanged and S is relinquished. This cycle is repeated until the time limit is reached.

In the early stages of this project, some C libraries (developed by L Corcoran, these libraries are available on the Internet) were used to implement PMX and Direct3. Here, the GA uses a generational model: an entire population is created at each iteration. In fact, this model works with two populations: the old one and the new one. The old population is used to generate the new population. Parents in the old population are selected for reproduction and they generate offspring in the new population. 30% of the old population is copied (or cloned) to the new population at each generation. This corresponds to a generational gap of 30%. A roulette selection technique was adopted where each member of the population is allocated a slice of a roulette wheel; the slice is proportional to the fitness of the chromosome.

Under this model, a GA might converge prematurely. Such a convergence may happen well before the ten minutes limit and when this situation occurs, the GA is stopped. Furthermore, this first model also suffers from some serious limitations in terms of computational efficiency and memory requirement. This prompted the decision to design a new GA. During the design of this new algorithm, the main emphasis was on the design of an efficient algorithm, i.e. this new model had to be *fast* and *memory efficient.*

The *second-generation GAs*, Direct3-s and Direct4-s, use a steady-state model. Initially, a set of n solutions is generated and sorted by fitness. The selection function works as follows. First, a number, x, is generated randomly in the range 1 to $n-1$. Then, a number y is generated, where $1 < y \leq x$ and the yth chromosome is selected for reproduction. Once two parents have been selected, the crossover operator generates one child which is then inserted at the correct position in the population and the weakest member of the population is deleted. This cycle of selection, recombination and replacement is repeated until the 10 minute limit is

reached.

Figure 6.1 shows that with this selection function fitter chromosomes have a much higher chance of being selected. To plot this graph, the selection function was run 10 million times. There were 100 chromosomes in the pool, ranked with respect to their cost from the best one to the worst one. The fittest chromosome was selected on 518692 occasions and the worst chromosome was only selected on 902 occasions.

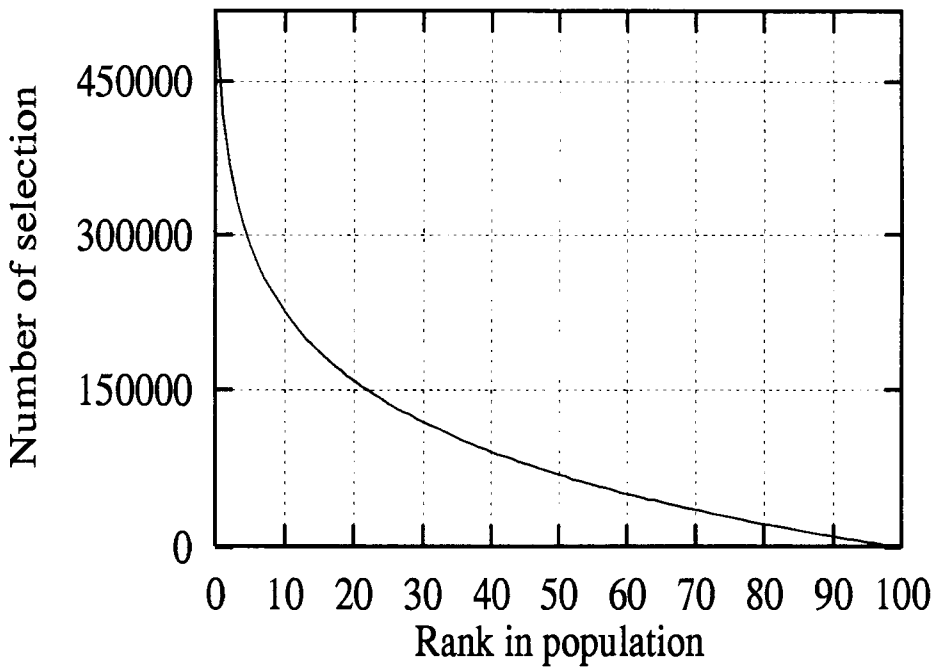

Figure 6.1: Selection function - an elitist strategy.

This selection strategy was adopted because the GA has a strict time-limit of 10 minutes. In this situation, this selection function provides a suitable balance between exploration of the solution space and rapid convergence.

Moreover, in order to prevent a premature convergence, the GA does not accept the insertion of duplicates in the population. Basically, a chromosome cannot be inserted if it is identical, in terms of cost, to a chromosome already present in the population. When this situation occurs, the chromosome is simply rejected and the algorithm generates a new chromosome. Finally, during these first experiments, the

four GAs have a population of 100 chromosomes and none of them uses a mutation operator.

Tables 6.1, 6.2, 6.3, 6.4 present the performances of the five algorithms over the full range of problems. A table entry gives the average value taken over the 5 applications of the algorithm to a given problem. There are five columns for each problem; each column corresponds to a different algorithm. Hence, if we scan across a row, we can compare algorithms on specific problems.

The data is analysed from 4 perspectives. Table 6.1 gives the average final cost reached by the five algorithms for the different problems. Table 6.2 gives the average amount of work done. Table 6.3 gives the average travel time. Table 6.4 gives the average amount of work not done.

Table 6.1 shows that, in general, the five algorithms find solutions with comparable costs, i.e. the solutions are in the same order of magnitude. This is a remarkable result as the cost function used in these tests does not have a smooth topology. Indeed, as discussed earlier, the cost of a solution is 600 times the duration of the jobs not done plus the different travel time and, thus even a marginal improvement in terms of work done leads to a significant reduction of cost. Despite this, the five algorithms generate (in most cases) solutions of similar cost. This signifies that, generally, a problem is equally difficult for the five algorithms. If one algorithm finds a given problem difficult, then all the others will find this particular problem difficult as well.

However, there are some noticeable exceptions. For instance, on the 40-2-0-3 problem, the average costs for RS, PMX, Direct3 and Direct3-s are 56334, 47417, 12963 and 12861 respectively but Direct4-s manages an average cost of only 1984.

This results from the fact that only Direct4-s can allocate all the work, while RS, PMX, Direct3 and Direct3-s still have 20, 75, 18 and 18 minutes respectively of work not done. As this example shows, massive differences (in terms of cost) can

be expected between two algorithms if one manages to allocate all the jobs and the other one does not. In this case, it can be said that the problem under study is easy for Direct4-s but hard for the other techniques. The situation where the different algorithms do not have the same behaviour only occurs in the 40-* problems with time-windows and technological constraints.

These differences in performance suggest that these particular problems are critical, i.e. some of the algorithms can allocate all the work, some cannot. Clearly, it appears that the load of the engineers is a key parameter that determines whether all the work is allocated or causes some to remain undone.

Typically, the 30-* problems are under-resourced and all the algorithms generate solutions with large costs. The best solution is simply the one which minimises the amount of work not done Travel is only a secondary objective here.

The 50-* problems are over-resourced and all the jobs are done. Hence, the reduction of travel becomes the primary concern and the best solution is the one that minimises travel. Since all jobs are allocated, travel offers the only scope for improvement.

For the 40-* problems in the middle of the spectrum, the situation is more delicate. Some problems have solutions in which all the work is done, and some have solutions in which some jobs remain unallocated. Clearly, these problems are critically resourced and they are the most interesting problems. It is of little interest to an enterprise to solve a problem by continually having excessive resources, or to continually fail to meet those objectives due to under-resourcing.

Tables 6.1 and 6.2 show that the solution with the largest amount of work done is always the best solution. This was guaranteed by the cost function. Furthermore, the best solution also tends to be the one which minimises travel. For instance, on the 30-0-0-1 problem, Direct4-s allocates the largest amount of work (12972) but it also has the smallest amount of travel (1286). Clearly, in this case, the best

solution is doing more work and less travel than the other solutions. This is a typical situation and suggests that reduction of travel is of prime importance. This observation will be used in Section 6.1.3 to create a repair algorithm.

Table 6.5 ranks the algorithms. Given a problem instance P_i, the performance of algorithm A_i is compared against the others. If an algorithm A_i finds a better cost than an algorithm A_j, then the algorithm A_i is awarded one point. If the two algorithms find solutions of equal value, then both get zero points. In total, an algorithm can score a maximum of four points and a minimum of 0 points for a given problem instance.

Figure 6.2 summarises the average, the minimum and the maximum scores for each algorithm over the entire set of 36 problems.

Algorithm	average	min	max
RS	0.42	0	1
PMX	0.58	0	1
Direct3	2.86	2	4
Direct3-s	2.97	2	4
Direct4-s	3.16	2	4

Figure 6.2: Ranking of GAs over 36 problems.

From Table 6.5 and Figure 6.2 , it can be seen that the three direct GAs totally dominate RS and the indirect candidate, PMX. In this particular set of experiments, there is no single example where PMX or RS provide a better solution than Direct3, Direct3-s or Direct4-s. Furthermore, in most cases, there is a significant gap between the solutions of RS and PMX, and the solutions of the three direct GAs. The average costs achieved by the PMX algorithm are remarkably similar to those achieved by RS; these two algorithms achieve average scores of 0.42 (for RS) and 0.58 (for PMX). These are very low scores compared with the scores of the three direct GAs. Clearly, the association of order-based GA and schedule-builder

is not suitable for this type of problem, i.e. the PMX algorithm does not guide its search properly.

Among the three direct GAs, there is no overall champion. Direct3, Direct3-s and Direct4-s have comparable average scores of 2.86, 2.97 and 3.16.

The GA using the knowledge-based crossover, Direct4-s, achieves the best score of 3.16. As expected, this crossover is especially effective on the 30-* problems and on the critical 40-* problems. On this particular set of problems, i.e. 12 under-resourced 30-* problems and 6 critical 40-* problems, Direct4-s reaches the best solution on 17 occasions and is only once beaten by Direct3-s. However, on the 50-* problems, Direct4-s is outperformed by Direct3 and Direct3-s; both of these GAs use the same random-based direct crossover.

These results come as no surprise. Direct4-s uses its intelligence to inherit the best tours during the crossover operation; in this case, the best tours are the ones which have the largest amount of work done. This knowledge-based crossover is particularly effective on under- or critically-resourced problems where the algorithm must use all its intelligence in order to reduce the amount of work not done. However, on the 50-* problems, where travel is the only scope for improvement, this technique does not guide the search properly and the two random-based direct GAs (Direct3 and Direct3-s) are more effective.

Direct3-s is particularly effective on the 50-* problems, where out of 12 instances it scores the maximum 4 points on 9 occasions. In general, Direct3-s is marginally better than Direct3. As these two algorithms use the same crossover, this difference suggests that, on average, the steady state model is more effective than the generational model.

Table 6.6 gives a guide to the amount of exploration performed by a given technique over a given problem in 10 minutes. The RS column shows the number of schedules created. The four other columns show the number of crossover operations. The two

algorithms, PMX and Direct3, experienced premature convergence in a number of instances, hence the two columns PMX and Direct3, also show (between brackets) the real amount of time in seconds spent by the algorithm before convergence.

In general, RS and PMX perform fewer iterations than the three direct GAs. Also, PMX performs slightly fewer iterations than RS. These results were expected. RS is a blind search, i.e. a new solution is generated and evaluated from scratch at each operation. These two operations are extremely expensive in terms of computational effort. The indirect GA, PMX, has the same behaviour, i.e. generation and evaluation from scratch for each crossover, but on top of this, PMX also has to generate order-based chromosomes and this small overhead explains the slightly bigger value of RS over PMX.

In the case of the direct GAs, the crossover operator combines two solutions so a direct GA only has to reschedule part of a solution. Therefore, the exploration of a single point in the search space will be substantially less expensive than for RS or PMX.

Looking at the RS column, it can be seen that more exploration takes place over problems with specialisation. Generally, RS examines 3 times as many points in the 30-*-3 problems as in the 30-*-0 problems. This also holds for the 40-* and the 50-* problems. PMX and the three direct GAs also experience this phenomenon.

Moving from under-resourced to over-resourced problems, direct GAs perform more exploration. This is because, in under-resourced problems, the computational cost of these GAs is dominated by the repair of chromosomes trying to insert as many jobs in the schedule as possible. In over-resourced problems, however, less repair is required and more exploration takes place.

Direct3-s always performs more iterations than Direct3. This is another indication that the steady-state model is more efficient than the generational model. Table 6.6 also shows that PMX and Direct3 converge extremely rapidly in some cases. For

instance, on the 40.2.3.1 problem, the two GAs converge and stop after only 202 and 56 seconds respectively. This suggests that the generational model is not capable of maintaining sufficient diversity within the population during the search. This might be due to the selection technique which gives too much emphasis to the fittest elements, but it might also be due to some other factors such as the population being too small, or the lack of a mutation operator.

Total convergence cannot happen under the steady state model. Indeed, this model does not accept duplicates and thus, the population cannot converge towards a uniform population. However, if the level of diversity is not kept at a proper level, the search might stagnate and the algorithm might well be spending most of its time creating solutions which will be rejected because they are already present in the population. This problem becomes more acute with large populations.

Table 6.7 shows the number of solutions rejected, in 10 minutes, by the two GAs, Direct3-s and Direct4-s. These are the only algorithms which use the steady state model. When compared with Table 6.6, which presents the number of solutions visited in 10 minutes, this table suggests that both Direct 3-s and Direct 4-s are spending most of their time creating solutions which are rejected immediately.

Algorithm	average	min	max
RS	-	-	-
PMX	-	-	-
Direct3	-	-	-
Direct3-s	42.5%	11%	83%
Direct4-s	27.6%	1%	78%

Figure 6.3: GAs - When the best solution was found (% of 10 minutes).

Table 6.8 shows, for each problem, when the best solution was found during the search, and Figure 6.3 summarises this information. Both tables show the consequence of the lack of diversity in the population. In most cases, the two GAs

find their best solutions relatively early during the search, i.e. they do not use the full 10 minutes, and this situation is even more acute when solving problems with tight technological constraints. This might be due to the fact that in these types of problems, the different GAs perform far more iterations, and therefore, the loss of diversity happens earlier in the 10 minutes.

Several options can be suggested in order to overcome this loss of diversity:

1. Incorporate a mutation operator. Here, a possible mutation would be to randomly move a job from one engineer to another. However, this type of action is already performed during the crossover operation by the internal repair algorithm, i.e. *implicit mutation*, and hence, this operator is not an option worth considering here.

2. Adopt a new selection function. Rather than using an exponential selection function such as the one displayed in Figure 6.1, one possible option would be to normalise the situation and to make the selection probability of each chromosome directly proportional to its ranking in the population, rather than to its individual cost. This normalisation technique would limit the risk of domination by one single super-individual. However, this technique was implemented in the generational model and did not improve the situation.

3. Increase the size of the population. Very small populations might not create enough variety. However, larger populations might be prohibitive in terms of computational effort and this must be considered in a time-constrained environment.

The following section will investigate this last option.

In conclusion, the direct GAs totally dominate PMX over the entire set of problems. In this particular study, the indirect GA is not directing its search properly and

appears to be just another form of random search. Crossover #3 (the random-based crossover) should be preferred for the solution of over-resourced problems and Crossover #4 (the knowledge-based crossover) should be preferred for both under-resourced and critical problems. The steady-state model is overall the best option.

		generational model		steady state model	
	RS	PMX	Direct3	Direct3-s	Direct4-s
30-0-0-1	975696	982923	690824	660190	588086
30-0-0-2	775956	768757	467641	426173	384688
30-0-0-3	617570	622975	293034	293033	251543
30-0-3-1	763409	765237	529049	516432	435282
30-0-3-2	815645	803022	584880	530766	487509
30-0-3-3	1092715	1078299	849329	824090	793437
30-2-0-1	909188	898370	786606	703645	649560
30-2-0-2	772461	767042	700241	651570	608226
30-2-0-3	1499448	1499493	1421860	1423532	1409089
30-2-3-1	1364732	1339495	1213344	1225979	1180816
30-2-3-2	1154086	1148643	1054976	993695	919782
30-2-3-3	909303	885900	781294	784890	765036
40-0-0-1	2246	2260	1367	1333	1335
40-0-0-2	2231	2236	1325	1335	1338
40-0-0-3	2186	2173	1299	1307	1322
40-0-3-1	2315	2298	1481	1555	1582
40-0-3-2	2200	2192	1476	1460	1527
40-0-3-3	2382	2405	1529	1562	1640
40-2-0-1	445075	454055	417804	387155	392486
40-2-0-2	54407	63443	54321	23583	5436
40-2-0-3	56334	47417	12963	12861	1984
40-2-3-1	22329	13401	2453	2273	2176
40-2-3-2	2279	2325	2083	1989	1954
40-2-3-3	191368	184196	142638	120974	108409
50-0-0-1	2313	2297	1415	1359	1374
50-0-0-2	2235	2233	1325	1402	1422
50-0-0-3	2234	2217	1317	1320	1280
50-0-3-1	2276	2277	1522	1575	1625
50-0-3-2	2365	2313	1521	1545	1654
50-0-3-3	2325	2317	1530	1610	1633
50-2-0-1	2184	2225	1684	1689	1776
50-2-0-2	2107	2101	1493	1560	1560
50-2-0-3	2119	2107	1549	1554	1581
50-2-3-1	2184	2189	1843	1867	1891
50-2-3-2	2354	2390	2112	2106	2049
50-2-3-3	2385	2416	1950	1988	2058

Table 6.1: GAs performance - Average final cost.

		generational model		steady state model	
	RS	PMX	Direct3	Direct3-s	Direct4-s
30-0-0-1	12327	12315	12801	12852	12972
30-0-0-2	12285	12297	12798	12867	12936
30-0-0-3	12279	12270	12819	12819	12888
30-0-3-1	12156	12153	12546	12567	12702
30-0-3-2	12144	12165	12528	12618	12690
30-0-3-3	12267	12291	12672	12714	12765
30-2-0-1	12183	12201	12387	12525	12615
30-2-0-2	12111	12120	12231	12312	12384
30-2-0-3	11889	11889	12018	12015	12039
30-2-3-1	11424	11466	11676	11655	11730
30-2-3-2	11835	11844	12000	12102	12225
30-2-3-3	11103	11142	11316	11310	11343
40-0-0-1	13155	13155	13155	13155	13155
40-0-0-2	12780	12780	12780	12780	12780
40-0-0-3	13410	13410	13410	13410	13410
40-0-3-1	12990	12990	12990	12990	12990
40-0-3-2	13575	13575	13575	13575	13575
40-0-3-3	13230	13230	13230	13230	13230
40-2-0-1	13287	13272	13332	13383	13374
40-2-0-2	13293	13278	13293	13344	13374
40-2-0-3	13560	13575	13632	13632	13650
40-2-3-1	13422	13437	13455	13455	13455
40-2-3-2	13635	13635	13635	13635	13635
40-2-3-3	13560	13572	13641	13677	13698
50-0-0-1	13860	13860	13860	13860	13860
50-0-0-2	13575	13575	13575	13575	13575
50-0-0-3	13215	13215	13215	13215	13215
50-0-3-1	13335	13335	13335	13335	13335
50-0-3-2	13740	13740	13740	13740	13740
50-0-3-3	13830	13830	13830	13830	13830
50-2-0-1	13185	13185	13185	13185	13185
50-2-0-2	13290	13290	13290	13290	13290
50-2-0-3	13710	13710	13710	13710	13710
50-2-3-1	13470	13470	13470	13470	13470
50-2-3-2	13215	13215	13215	13215	13215
50-2-3-3	13335	13335	13335	13335	13335

Table 6.2: GAs performance - Average amount of work done in minutes.

		generational model		steady state model	
	RS	PMX	Direct3	Direct3-s	Direct4-s
30-0-0-1	1896	1923	1424	1390	1286
30-0-0-2	1956	1957	1441	1373	1288
30-0-0-3	1970	1975	1434	1433	1343
30-0-3-1	2009	2037	1649	1632	1482
30-0-3-2	2045	2022	1680	1566	1509
30-0-3-3	1915	1899	1529	1490	1437
30-2-0-1	1988	1970	1806	1645	1560
30-2-0-2	2061	2042	1841	1770	1626
30-2-0-3	1848	1893	1660	1532	1489
30-2-3-1	2132	2095	1944	1979	1816
30-2-3-2	2086	2043	1976	1895	1782
30-2-3-3	2103	2100	1894	1890	1836
40-0-0-1	2246	2260	1367	1333	1335
40-0-0-2	2231	2236	1325	1335	1338
40-0-0-3	2186	2173	1299	1307	1322
40-0-3-1	2315	2298	1481	1555	1582
40-0-3-2	2200	2192	1476	1460	1527
40-0-3-3	2382	2405	1529	1562	1640
40-2-0-1	2275	2255	2004	1955	1886
40-2-0-2	2207	2243	2121	1983	1836
40-2-0-3	2334	2417	2163	2061	1984
40-2-3-1	2529	2601	2453	2273	2176
40-2-3-2	2279	2325	2083	1989	1954
40-2-3-3	2368	2396	2238	2174	2209
50-0-0-1	2313	2297	1415	1359	1374
50-0-0-2	2235	2233	1325	1402	1422
50-0-0-3	2234	2217	1317	1320	1280
50-0-3-1	2276	2277	1522	1575	1625
50-0-3-2	2365	2313	1521	1545	1654
50-0-3-3	2325	2317	1530	1610	1633
50-2-0-1	2184	2225	1684	1689	1776
50-2-0-2	2107	2101	1493	1560	1560
50-2-0-3	2119	2107	1549	1554	1581
50-2-3-1	2184	2189	1843	1867	1891
50-2-3-2	2354	2390	2112	2106	2049
50-2-3-3	2385	2416	1950	1988	2058

Table 6.3: GAs performance - Average amount of travel time in minutes.

		generational model		steady state model	
	RS	PMX	Direct3	Direct3-s	Direct4-s
30-0-0-1	1623	1635	1149	1098	978
30-0-0-2	1290	1278	777	708	639
30-0-0-3	1026	1035	486	486	417
30-0-3-1	1269	1272	879	858	723
30-0-3-2	1356	1335	972	882	810
30-0-3-3	1818	1794	1413	1371	1320
30-2-0-1	1512	1494	1308	1170	1080
30-2-0-2	1284	1275	1164	1083	1011
30-2-0-3	2496	2496	2367	2370	2346
30-2-3-1	2271	2229	2019	2040	1965
30-2-3-2	1920	1911	1755	1653	1530
30-2-3-3	1512	1473	1299	1305	1272
40-0-0-1	0	0	0	0	0
40-0-0-2	0	0	0	0	0
40-0-0-3	0	0	0	0	0
40-0-3-1	0	0	0	0	0
40-0-3-2	0	0	0	0	0
40-0-3-3	0	0	0	0	0
40-2-0-1	738	753	693	642	651
40-2-0-2	87	102	87	36	6
40-2-0-3	90	75	18	18	0
40-2-3-1	33	18	0	0	0
40-2-3-2	0	0	0	0	0
40-2-3-3	315	303	234	198	177
50-0-0-1	0	0	0	0	0
50-0-0-2	0	0	0	0	0
50-0-0-3	0	0	0	0	0
50-0-3-1	0	0	0	0	0
50-0-3-2	0	0	0	0	0
50-0-3-3	0	0	0	0	0
50-2-0-1	0	0	0	0	0
50-2-0-2	0	0	0	0	0
50-2-0-3	0	0	0	0	0
50-2-3-1	0	0	0	0	0
50-2-3-2	0	0	0	0	0
50-2-3-3	0	0	0	0	0

Table 6.4: GAs performance - Average amount of work not done in minutes.

		generational model		steady state model	
	RS	PMX	Direct3	Direct3-s	Direct4-s
30-0-0-1	1	0	2	3	4
30-0-0-2	0	1	2	3	4
30-0-0-3	1	0	2	3	4
30-0-3-1	1	0	2	3	4
30-0-3-2	0	1	2	3	4
30-0-3-3	0	1	2	3	4
30-2-0-1	0	1	2	3	4
30-2-0-2	0	1	2	3	4
30-2-0-3	1	0	3	2	4
30-2-3-1	0	1	3	2	4
30-2-3-2	0	1	2	3	4
30-2-3-3	0	1	3	2	4
40-0-0-1	1	0	2	4	3
40-0-0-2	1	0	4	3	2
40-0-0-3	0	1	4	3	2
40-0-3-1	0	1	4	3	2
40-0-3-2	0	1	3	4	2
40-0-3-3	1	0	4	3	2
40-2-0-1	1	0	2	4	3
40-2-0-2	1	0	2	3	4
40-2-0-3	0	1	2	3	4
40-2-3-1	0	1	2	3	4
40-2-3-2	1	0	2	3	4
40-2-3-3	0	1	2	3	4
50-0-0-1	0	1	2	4	3
50-0-0-2	0	1	4	3	2
50-0-0-3	0	1	3	2	4
50-0-3-1	1	0	4	3	2
50-0-3-2	0	1	4	3	2
50-0-3-3	0	1	4	3	2
50-2-0-1	1	0	4	3	2
50-2-0-2	0	1	4	2	3
50-2-0-3	0	1	4	3	2
50-2-3-1	1	0	4	3	2
50-2-3-2	1	0	2	3	4
50-2-3-3	1	0	4	3	2

Table 6.5: GAs performance - Ranking of GAs (versus cost).

		generational model		steady state model	
	RS	PMX	Direct3	Direct3-s	Direct4-s
30-0-0-1	1400	1292 (616)	5603 (602)	8190	9700
30-0-0-2	1400	1292 (615)	6296 (570)	11100	13830
30-0-0-3	1400	1292 (618)	7548 (552)	14990	18880
30-0-3-1	4400	4175 (603)	5657 (179)	36560	42050
30-0-3-2	4400	3944 (572)	5032 (169)	36210	38900
30-0-3-3	4600	4352 (601)	6337 (234)	26640	29130
30-2-0-1	1400	1346 (620)	5086 (529)	7460	7970
30-2-0-2	1420	1360 (622)	5344 (551)	7400	8590
30-2-0-3	1500	1428 (616)	4188 (603)	4820	4910
30-2-3-1	4680	4420 (605)	5208 (221)	18360	17650
30-2-3-2	4800	3903 (520)	4120 (163)	21830	24470
30-2-3-3	5200	4896 (604)	5820 (194)	24030	25660
40-0-0-1	1400	1292 (623)	7656 (321)	64000	69340
40-0-0-2	1400	1292 (623)	8690 (346)	60500	60270
40-0-0-3	1400	1292 (625)	7765 (334)	7014C	50050
40-0-3-1	4500	4243 (605)	7969 (149)	157340	113300
40-0-3-2	4500	4243 (603)	7629 (151)	149060	113670
40-0-3-3	4700	4080 (552)	8309 (147)	171480	130280
40-2-0-1	1500	1360 (608)	6174 (364)	15160	15340
40-2-0-2	1500	1278 (576)	3427 (133)	81730	78210
40-2-0-3	1500	1237 (560)	3998 (117)	91110	82450
40-2-3-1	4600	1373 (202)	3359 (56)	186310	127170
40-2-3-2	4600	3576 (498)	4896 (74)	177130	154750
40-2-3-3	4200	3359 (518)	3930 (89)	85420	87750
50-0-0-1	1400	1292 (626)	8187 (330)	66540	65010
50-0-0-2	1400	1278 (620)	8826 (352)	73420	64250
50-0-0-3	1400	1292 (624)	8064 (306)	75420	49720
50-0-3-1	4200	4025 (606)	6963 (138)	142350	111230
50-0-3-2	4300	4080 (604)	7534 (146)	143190	125800
50-0-3-3	4400	4148 (604)	7316 (145)	163200	114920
50-2-0-1	1500	1360 (611)	7956 (281)	67400	75030
50-2-0-2	1400	1319 (610)	8078 (297)	74340	72610
50-2-0-3	1500	1360 (614)	8404 (312)	66440	78000
50-2-3-1	4500	4066 (584)	6596 (122)	174430	116160
50-2-3-2	4000	3753 (605)	5521 (96)	164700	129830
50-2-3-3	4800	4528 (603)	6922 (121)	172670	130680

Table 6.6: GAs performance - Amount of iterations in 10 minutes.

		generational model		steady state model	
	RS	PMX	Direct3	Direct3-s	Direct4-s
30-0-0-1	-	-	-	5267	7599
30-0-0-2	-	-	-	8046	11174
30-0-0-3	-	-	-	11013	15438
30-0-3-1	-	-	-	31190	29854
30-0-3-2	-	-	-	27821	30805
30-0-3-3	-	-	-	22864	25715
30-2-0-1	-	-	-	4703	6079
30-2-0-2	-	-	-	4343	6693
30-2-0-3	-	-	-	3287	3968
30-2-3-1	-	-	-	16256	15049
30-2-3-2	-	-	-	18238	20237
30-2-3-3	-	-	-	21745	22994
40-0-0-1	-	-	-	62868	68570
40-0-0-2	-	-	-	59203	59222
40-0-0-3	-	-	-	69049	49167
40-0-3-1	-	-	-	155762	111778
40-0-3-2	-	-	-	146885	111803
40-0-3-3	-	-	-	169267	128575
40-2-0-1	-	-	-	13719	14178
40-2-0-2	-	-	-	78269	75347
40-2-0-3	-	-	-	87596	80357
40-2-3-1	-	-	-	184238	122417
40-2-3-2	-	-	-	171470	152128
40-2-3-3	-	-	-	82859	82768
50-0-0-1	-	-	-	65246	63942
50-0-0-2	-	-	-	72024	62893
50-0-0-3	-	-	-	74018	48799
50-0-3-1	-	-	-	141087	110428
50-0-3-2	-	-	-	141676	124813
50-0-3-3	-	-	-	161978	113756
50-2-0-1	-	-	-	66178	74005
50-2-0-2	-	-	-	73108	71386
50-2-0-3	-	-	-	65288	77084
50-2-3-1	-	-	-	172891	114196
50-2-3-2	-	-	-	160497	126058
50-2-3-3	-	-	-	170053	129504

Table 6.7: GAs performance - Amount of solutions rejected in 10 minutes.

		generational model		steady state model	
	RS	PMX	Direct3	Direct3-s	Direct4-s
30-0-0-1	-	-	-	74	78
30-0-0-2	-	-	-	37	39
30-0-0-3	-	-	-	44	37
30-0-3-1	-	-	-	23	9
30-0-3-2	-	-	-	11	19
30-0-3-3	-	-	-	22	22
30-2-0-1	-	-	-	58	41
30-2-0-2	-	-	-	67	46
30-2-0-3	-	-	-	83	65
30-2-3-1	-	-	-	36	23
30-2-3-2	-	-	-	39	8
30-2-3-3	-	-	-	25	23
40-0-0-1	-	-	-	45	24
40-0-0-2	-	-	-	54	12
40-0-0-3	-	-	-	55	35
40-0-3-1	-	-	-	50	42
40-0-3-2	-	-	-	34	40
40-0-3-3	-	-	-	25	12
40-2-0-1	-	-	-	50	33
40-2-0-2	-	-	-	66	18
40-2-0-3	-	-	-	50	27
40-2-3-1	-	-	-	24	41
40-2-3-2	-	-	-	15	1
40-2-3-3	-	-	-	39	14
50-0-0-1	-	-	-	48	41
50-0-0-2	-	-	-	47	17
50-0-0-3	-	-	-	63	24
50-0-3-1	-	-	-	30	28
50-0-3-2	-	-	-	46	24
50-0-3-3	-	-	-	42	34
50-2-0-1	-	-	-	56	8
50-2-0-2	-	-	-	34	30
50-2-0-3	-	-	-	55	62
50-2-3-1	-	-	-	27	6
50-2-3-2	-	-	-	28	5
50-2-3-3	-	-	-	27	6

Table 6.8: GAs performance - When the best solution was found (% of 10 minutes).

6.1.2 The effect of population size.

This section investigates the effects of different population sizes on the performance of GAs. The principle objective here is to reduce the loss of diversity by increasing the size of the population. Only one single direct GA is studied and, again, RS is used as the reference algorithm. Direct4-s has been chosen for these tests because this algorithm achieves the best performance on the hard problems where some work remains undone, i.e. under-resourced 30-* and critical 40-* problems.

This algorithm was run with three different population sizes: 100, 200 and 400 chromosomes. Therefore, the performance of four different runs are reported in the next tables: RS, Direct4-s:100, Direct4-s:200 and Direct4-s:400.

Analysis of the results in the previous section show that there is no instance where work is being traded for travel, i.e. a low cost solution is always better than a high cost solution (never does less work). Consequently, work done, travel or work not done are no longer reported. Hence, the data is only analysed from one perspective: final cost. Table 6.9 gives the average final cost reached by the algorithms on the different problems.

Direct4-s:200 always performs better than Direct4-s:100 and, in general, Direct4-s:400 performs better than Direct4-s:200. However, there are some noticeable exceptions where Direct4-s:200 achieves better solutions than Direct4-s:400. For instance, on problems with no technological or temporal constraints, i.e. 30-0-0-*, 40-0-0-* and 50-0-0-*, Direct4-s:200 always performs significantly better. On the last two groups (40-0-0-* and 50-0-0-*), even the "smallest" GA, Direct4-s:100, is reaching far better solutions than direct4-s:400. Intriguingly, in these two examples, the performances of Direct4-s:400 are comparable to the performances of RS.

An explanation for these peculiar results can be found in Table 6.10 which shows the amount of exploration performed by the four algorithms in 10 minutes. As

expected, Direct4-s:100 performs more iterations than Direct4-s:200 which in turn performs more iterations than Direct4-s:400. Clearly, larger populations require more time to create the initial population, and thus, less time is available for the actual search.

However, a linear relationship should be expected, such that if the population size is doubled, the time required to create the initial population should double as well. Unfortunately, in some cases, the relationship seems to be much worse than linear. In some instances, 10 minutes are not even sufficient for Direct4-s:400 to create its initial population. For instance, this is the case for all the 40-0-0-* and all the 50-0-0-* problems. Figure 6.4 shows the time in seconds that the schedule-builder needs in order to create an initial population free of duplicates for the 30-0-0-1 problem. The situation is even more acute for any of the 40-0-0-* or 50-0-0-* problems.

The last point *"free of duplicates"* is the main problem here. If duplicates were allowed, then the relationship would be linear. However, in order to limit the risk of premature convergence, duplicates are not accepted, and as the search goes on, it seems harder and harder for the schedule-builder to create a new, not yet visited, solution and this explains the exponential behaviour. After a certain stage, the schedule-builder is spending most of its time generating solutions which are rejected immediately.

Table 6.11 illustrates this problem; this table reports the percentage of schedules accepted when 100, 500, 1000, 2000 or 5000 schedules are created. For instance, when the schedule-builder creates 100 solutions for a 30-0-0-1 problem, 98% are accepted and 2% are rejected because they are duplicates. when the schedule-builder creates 5000 schedules only 33.9%, i.e. 1693 schedules are accepted.

Table 6.11 shows that the situation is particularly serious for problems with no constraints, i.e. 30-0-0-*, 40-0-0-*, and 50-0-0-* problems, and also for most of the 50-* problems with the noticeable exception of the 50-2-3-* problems.

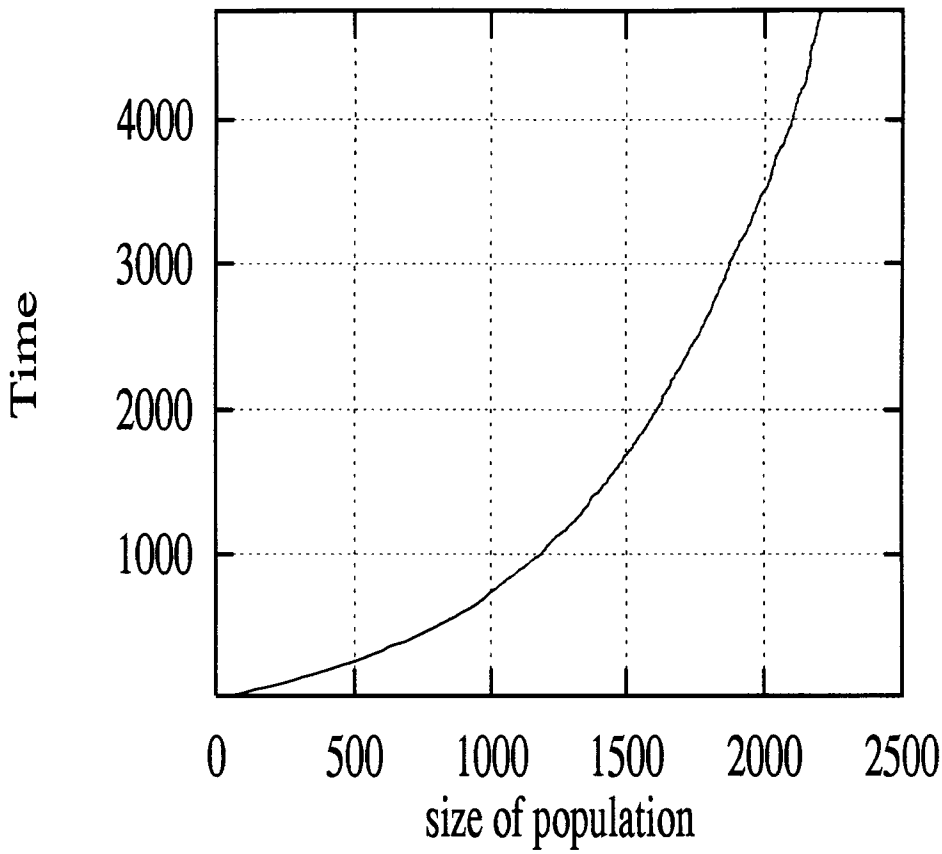

Figure 6.4: Population free of duplicates: the larger, the harder.

This phenomenon raises an important issue. It suggests a negative answer to the question "Could a GA benefit from a *smart* start?" In other words, if a GA uses an intelligent schedule-builder to generate its initial population, will the algorithm find better solutions?

For instance, several methods could be envisaged for the development of an intelligent schedule-builder:

- Jobs could be allocated to engineers in a round-robin manner, resulting in a more equitable distribution of jobs to engineers.
- Geographical information could be taken into account, such that tours radiate from the base, resulting in natural clusters of jobs.

- Some look-ahead techniques might be employed such that critical jobs and engineers can be identified and scheduled before all others.

In this study, a rather naive schedule-builder was used. As seen earlier, it takes an ordered list of engineers and allocate jobs to each engineer in turn. During this operation, not a single heuristic is used. However, this basic schedule-builder is already generating a large amount of duplicates and unfortunately, one should expect a more intelligent schedule-builder to be even more deterministic [4] and generate even more duplicates.

In these circumstances, it seems unlikely that a GA could benefit from a good start, i.e. if the *smart* schedule-builder is too deterministic. Clearly, with an intelligent schedule-builder, the different solution-candidates in the initial population tend to be similar - good quality solutions but similar solutions. The GA may converge rapidly towards a sub-optimal solution, thus losing the benefits of a good starting point. Obviously, the best option would be an intelligent schedule-builder which creates radically different solutions and thus maintains the level of diversity in the population.

Nevertheless, there might be the possibility of seeding the initial population; i.e, only a limited section of the population is generated by the intelligent method, the rest (say 80% for instance) is generated by the normal, i.e. naive, method. This approach might provide the best compromise, since the GA would have access to a limited set of good quality solutions but at the same time, the population would have a sufficient level of diversity.

In conclusion, there are two conflicting observations. First, the presence of duplicates in the population has to be avoided because it leads to a loss of diversity which, in turn, leads to a premature convergence towards a sub-optimal solution.

[4] Ultimately, the *"best"* schedule-builder would always generate the same solution, i.e. this solution would be the "optimal" solution.

On the other hand, if duplicates are rejected, then the GA might spend most of its time generating solutions which are rejected straight away.

In consequence, a GA might improve its performances by accepting duplicates despite the possible loss of diversity.

Two GAs which accept duplicates are studied in the remainder of this section:

1. The first GA only accepts duplicates when the initial population is being created, but rejects then during the rest of the search. This GA will be referred to as Dupli-1.

2. The second GA accepts duplicates at any time during the search. This GA will be referred to as Dupli-2.

Both Dupli-1 and Dupli-2 use the steady-state model and Crossover #4, they also both have a population of 400 chromosomes. Hence, direct comparisons with Direct4-s:100, Direct4-s:200 and Direct4-s:400 are possible.

Table 6.12 gives the average final cost reached by RS, Direct4-s:100, Direct4-s:200, Direct4-s:400, Dupli-1 and Dupli-2. Table 6.13 gives the ranking of these algorithms for the different problems and Figure 6.5 summarises this information. It gives the average score, the minimum and the maximum score for each algorithm over the entire set of problems; in this case, each algorithm can score a minimum of 0 point and a maximum of 5 points.

On average, Dupli-1 and Dupli-2 find the best solutions and, hence, have the best scores of all the algorithms: 4.05 for Dupli-1 and 3.05 for Dupli-2. Dupli-1 seems to be a more robust algorithm.

Of the 50-* problems, the 50-2-3-1 problem is the only instance (out of 12) where the best solution is not found by either Dupli-1 or Dupli-2. In the same manner, on the easy 40-* problems (40-* problems where all the work is done), the best

Algorithm	average	min	max
R S	0.0	0	0
Direct4-s:100	1.27	1	2
Direct4-s:200	3.11	2	5
Direct4-s:400	3.05	1	5
Dupli-1	4.05	2	5
Dupli-2	3.5	1	5

Figure 6.5: Different population sizes, different ranking.

solution is typically found either by Dupli-1 or by Dupli-2 as well. Therefore, *"accept duplicates"* appears to be a good policy on over-resourced problems.

The situation is more delicate for under-resourced problems. For instance, on the 30-* problems, Dupli-1 and Dupli-2 together can only manage 4 *pole-positions.* Generally, Dupli-2 is performing rather poorly on these problems. Direct4-s:200 is particularly effective on problems with no technological constraints and Direct4-s:400 works well on problems with tight technological constraints.

Table 6.7 (in the previous section) showed that the level of rejection was especially important in the case of over-resourced problems. In these situations, the GA was spending most of its search rejecting schedules. By accepting duplicates, the GA converges more rapidly, but at least, the algorithm is moving around in the search space. The level of rejection is not so important in under-resourced problems and these *"accept duplicates"* policies might not be desirable.

Table 6.14 shows (for the 6 algorithms) when the best solutions were found (in %) on the different problems. Observe that the results for Dupli-2 might be slightly misleading. Dupli-2 accepts duplicates at any time during the search, hence there can be several copies of the best solution in the population. In the worst case, if the population has converged, each member of the population is a copy of the best solution. Despite this, this table only indicates when the last copy of the best solution was created.

Table 6.14 shows that, typically, when the population size is increased, the algorithm finds its best solution at a later stage. This behaviour was expected. First, the generation of a larger initial population requires more computational effort, so the genetic search starts later. More importantly, a larger population helps to delay premature convergence and hence, the algorithm can use the extra-time to produce better solutions.

Dupli-1 exhibits a typical behaviour, it finds its best solutions earlier when solving problems with tight technological constraints. On problems with no such constraints, Dupli-1 finds its best solutions in the latter stages of the search, i.e. after 90-95% of the search. This suggests that this algorithm might benefit from longer runs; there is still scope for significant improvement at the end of the 10 minutes.

In conclusion, larger populations should provide a higher level of variety in the pool and, hence, help the GA to reach better solutions by preventing premature convergence. However, the limitations of the schedule-builder mean that the algorithm spends a prohibitive amount of time creating large populations unless duplicates are allowed in the population. This study also clearly suggests that a GA might not be able to benefit from a *smart* schedule-builder which affects the level of diversity in the population.

	RS	Direct4-s:100	Direct4-s:200	Direct4-s:400
30-0-0-1	975696	588086	526785	553823
30-0-0-2	775956	384688	319818	325189
30-0-0-3	617570	251543	175831	186656
30-0-3-1	763409	435282	374021	325324
30-0-3-2	815645	487509	451450	354138
30-0-3-3	1092715	793437	712334	651047
30-2-0-1	909188	649560	582873	642311
30-2-0-2	772461	608226	554164	584830
30-2-0-3	1499448	1409089	1380200	1396575
30-2-3-1	1364732	1180816	1123114	1083411
30-2-3-2	1154086	919782	851285	804407
30-2-3-3	909303	765036	750542	654992
40-0-0-1	2246	1335	1249	2201
40-0-0-2	2231	1338	1256	2217
40-0-0-3	2186	1322	1215	2157
40-0-3-1	2315	1582	1450	1406
40-0-3-2	2200	1527	1394	1338
40-0-3-3	2382	1640	1512	1495
40-2-0-1	445075	392486	370864	352839
40-2-0-2	54407	5436	1750	1841
40-2-0-3	56334	1984	1717	1719
40-2-3-1	22329	2176	1960	1860
40-2-3-2	2279	1954	1838	1708
40-2-3-3	191368	108409	46982	9122
50-0-0-1	2313	1374	1276	2276
50-0-0-2	2235	1422	1252	2189
50-0-0-3	2234	1280	1210	2183
50-0-3-1	2276	1625	1513	1465
50-0-3-2	2365	1654	1519	1469
50-0-3-3	2325	1633	1477	1452
50-2-0-1	2184	1776	1628	2165
50-2-0-2	2107	1560	1487	2049
50-2-0-3	2119	1581	1498	2075
50-2-3-1	2184	1891	1771	1642
50-2-3-2	2354	2049	1965	1876
50-2-3-3	2385	2058	1882	1835

Table 6.9: Average final cost for different population sizes.

	RS	Direct4-s:100	Direct4-s:200	Direct4-s:400
30-0-0-1	1400	9700	8150	4170
30-0-0-2	1400	13830	11450	4410
30-0-0-3	1400	18880	15900	6030
30-0-3-1	4400	42050	37800	32960
30-0-3-2	4400	38900	38340	30450
30-0-3-3	4600	29130	27140	22470
30-2-0-1	1400	7970	6660	3570
30-2-0-2	1420	8590	6970	3700
30-2-0-3	1500	4910	4100	2930
30-2-3-1	4680	17650	18470	15940
30-2-3-2	4800	24470	20830	19410
30-2-3-3	5200	25660	22380	21280
40-0-0-1	1400	69340	27660	0
40-0-0-2	1400	60270	25080	0
40-0-0-3	1400	50050	26550	0
40-0-3-1	4500	113300	89390	45640
40-0-3-2	4500	113670	85160	44770
40-0-3-3	4700	130280	83880	47090
40-2-0-1	1500	15340	11950	5930
40-2-0-2	1500	78210	54100	7570
40-2-0-3	1500	82450	42580	13220
40-2-3-1	4600	127170	99340	59430
40-2-3-2	4600	154750	92290	50350
40-2-3-3	4200	87750	80080	71420
50-0-0-1	1400	65010	30140	0
50-0-0-2	1400	64250	25650	0
50-0-0-3	1400	49720	25570	0
50-0-3-1	4200	111230	76650	28260
50-0-3-2	4300	125800	81400	34650
50-0-3-3	4400	114920	76610	34730
50-2-0-1	1500	75030	41920	0
50-2-0-2	1400	72610	31950	0
50-2-0-3	1500	78000	30160	0
50-2-3-1	4500	116160	104020	55650
50-2-3-2	4000	129830	99450	55770
50-2-3-3	4800	130680	93960	54070

Table 6.10: Amount of iterations for different population sizes.

	100	500	1000	2000	5000
30-0-0-1	98	82.4	70.4	55.15	33.86
30-0-0-2	97	83.4	71.2	54.55	34.2
30-0-0-3	95	83	71.6	55.3	33.86
30-0-3-1	99	92.4	88.1	79.15	61.66
30-0-3-2	99	92.4	89.2	79.05	60.64
30-0-3-3	98	91.6	85.5	76.35	55.58
30-2-0-1	99	90.2	84.1	72.8	50.7
30-2-0-2	96	89.6	82.7	70.5	49.92
30-2-0-3	98	90.4	84	72.7	52.92
30-2-3-1	99	96.6	94.8	90.55	78.42
30-2-3-2	99	97.8	94.7	90	77.06
30-2-3-3	98	96.2	94.3	89.05	77.26
40-0-0-1	82	45.4	28.3	16.25	7.42
40-0-0-2	81	48.4	29.8	17.4	7.96
40-0-0-3	84	45.2	27.9	15.8	7.48
40-0-3-1	97	87	79.5	68.1	50.94
40-0-3-2	97	91.6	85.5	74.5	56.64
40-0-3-3	96	90.6	84.4	73.8	56.34
40-2-0-1	99	95.8	90.7	84.5	69.34
40-2-0-2	99	94.4	90	81.15	62.8
40-2-0-3	97	95.2	91	84.45	68.9
40-2-3-1	98	96.8	94.6	89.9	78.4
40-2-3-2	97	97	94.2	87.4	75.42
40-2-3-3	99	97	93.8	88.25	73.9
50-0-0-1	84	45	28	16.7	8.04
50-0-0-2	88	46.8	29.3	17.7	8
50-0-0-3	84	45	29.1	16.5	7.48
50-0-3-1	86	50	32.3	20	10.96
50-0-3-2	85	52.2	35.6	22.4	13.3
50-0-3-3	87	53.8	35.9	23.25	13.26
50-2-0-1	85	47.8	27.9	16.25	7.7
50-2-0-2	81	44.6	27.9	16.5	7.74
50-2-0-3	79	43	27	15.7	7.12
50-2-3-1	97	90.6	84.1	71.4	52.24
50-2-3-2	98	95.4	93.2	84.5	67.24
50-2-3-3	98	90.8	84.3	73.95	56.14

Table 6.11: % of chromosomes accepted for different population sizes.

	RS	Direct4-s:100	Direct4-s:200	Direct4-s:400	Dupli-1	Dupli-2
30-0-0-1	975696	588086	526785	553823	535779	559222
30-0-0-2	775956	384688	319818	325189	308984	341434
30-0-0-3	617570	251543	175831	186656	193840	201057
30-0-3-1	763409	435282	374021	325324	301907	413671
30-0-3-2	815645	487509	451450	354138	379346	438829
30-0-3-3	1092715	793437	712334	651047	690690	744747
30-2-0-1	909188	649560	582873	642311	645932	636926
30-2-0-2	772461	608226	554164	584830	577598	550580
30-2-0-3	1499448	1409089	1380200	1396575	1387475	1387506
30-2-3-1	1364732	1180816	1123114	1083411	1088868	1126788
30-2-3-2	1154086	919782	851285	804407	802645	854868
30-2-3-3	909303	765036	750542	654992	691023	701913
40-0-0-1	2246	1335	1249	2201	1240	1150
40-0-0-2	2231	1338	1256	2217	1199	1123
40-0-0-3	2186	1322	1215	2157	1212	1135
40-0-3-1	2315	1582	1450	1406	1416	1402
40-0-3-2	2200	1527	1394	1338	1329	1397
40-0-3-3	2382	1640	1512	1495	1458	1424
40-2-0-1	445075	392486	370864	352839	345682	342030
40-2-0-2	54407	5436	1750	1841	1824	9033
40-2-0-3	56334	1984	1717	1719	1739	1832
40-2-3-1	22329	2176	1960	1860	1845	2042
40-2-3-2	2279	1954	1838	1708	1691	1839
40-2-3-3	191368	108409	46982	9122	27135	61440
50-0-0-1	2313	1374	1276	2276	1289	1139
50-0-0-2	2235	1422	1252	2189	1252	1174
50-0-0-3	2234	1280	1210	2183	1182	1115
50-0-3-1	2276	1625	1513	1465	1447	1423
50-0-3-2	2365	1654	1519	1469	1420	1443
50-0-3-3	2325	1633	1477	1452	1443	1442
50-2-0-1	2184	1776	1628	2165	1589	1527
50-2-0-2	2107	1560	1487	2049	1479	1352
50-2-0-3	2119	1581	1498	2075	1454	1436
50-2-3-1	2184	1891	1771	1642	1700	1725
50-2-3-2	2354	2049	1965	1876	1836	1912
50-2-3-3	2385	2058	1882	1835	1789	1851

Table 6.12: Average final cost for different population sizes and acceptance policies.

	RS	Direct4-s:100	Direct4-s:200	Direct4-s:400	Dupli-1	Dupli-2
30-0-0-1	0	1	5	3	4	2
30-0-0-2	0	1	4	3	5	2
30-0-0-3	0	1	5	4	3	2
30-0-3-1	0	1	3	4	5	2
30-0-3-2	0	1	2	5	4	3
30-0-3-3	0	1	3	5	4	2
30-2-0-1	0	1	5	3	2	4
30-2-0-2	0	1	4	2	3	5
30-2-0-3	0	1	5	2	4	3
30-2-3-1	0	1	3	5	4	2
30-2-3-2	0	1	3	4	5	2
30-2-3-3	0	1	2	5	4	3
40-0-0-1	0	2	3	1	4	5
40-0-0-2	0	2	3	1	4	5
40-0-0-3	0	2	3	1	4	5
40-0-3-1	0	1	2	4	3	5
40-0-3-2	0	1	3	4	5	2
40-0-3-3	0	1	2	3	4	5
40-2-0-1	0	1	2	3	4	5
40-2-0-2	0	2	5	3	4	1
40-2-0-3	0	1	5	4	3	2
40-2-3-1	0	1	3	4	5	2
40-2-3-2	0	1	3	4	5	2
40-2-3-3	0	1	3	5	4	2
50-0-0-1	0	2	4	1	3	5
50-0-0-2	0	2	3	1	4	5
50-0-0-3	0	2	3	1	4	5
50-0-3-1	0	1	2	3	4	5
50-0-3-2	0	1	2	3	5	4
50-0-3-3	0	1	2	3	4	5
50-2-0-1	0	2	3	1	4	5
50-2-0-2	0	2	3	1	4	5
50-2-0-3	0	2	3	1	4	5
50-2-3-1	0	1	2	5	4	3
50-2-3-2	0	1	2	4	5	3
50-2-3-3	0	1	2	4	5	3

Table 6.13: Ranking for different population sizes and acceptance policies.

	RS	Direct4-s:100	Direct4-s:200	Direct4-s:400	Dupli-1	Dupli-2
30-0-0-1	-	78	62	91	96	99
30-0-0-2	-	39	54	94	91	99
30-0-0-3	-	37	76	92	97	99
30-0-3-1	-	9	61	63	64	99
30-0-3-2	-	19	40	43	67	99
30-0-3-3	-	22	41	79	62	99
30-2-0-1	-	41	89	95	93	97
30-2-0-2	-	46	80	92	94	98
30-2-0-3	-	65	82	95	92	98
30-2-3-1	-	23	52	82	73	83
30-2-3-2	-	8	64	78	62	99
30-2-3-3	-	23	63	74	90	99
40-0-0-1	-	24	42	0	94	99
40-0-0-2	-	12	64	0	97	99
40-0-0-3	-	35	59	0	90	99
40-0-3-1	-	42	45	66	73	99
40-0-3-2	-	40	34	64	56	99
40-0-3-3	-	12	34	63	67	99
40-2-0-1	-	33	73	88	96	99
40-2-0-2	-	18	41	94	88	99
40-2-0-3	-	27	48	86	96	99
40-2-3-1	-	41	27	74	76	99
40-2-3-2	-	1	24	67	52	99
40-2-3-3	-	14	18	64	64	99
50-0-0-1	-	41	58	0	81	99
50-0-0-2	-	17	53	0	94	99
50-0-0-3	-	24	40	0	93	99
50-0-3-1	-	28	47	67	75	99
50-0-3-2	-	24	56	68	68	99
50-0-3-3	-	34	35	67	74	99
50-2-0-1	-	8	48	0	87	99
50-2-0-2	-	30	44	0	94	99
50-2-0-3	-	62	39	0	84	99
50-2-3-1	-	6	46	62	55	99
50-2-3-2	-	5	47	59	59	99
50-2-3-3	-	6	47	71	65	99

Table 6.14: When the best solution is found (in % of 10 minutes) for different population sizes and acceptance policies.

6.1.3 GA with repair algorithm.

In this section, a GA is coupled with a repair algorithm [5]. In their original form, GAs are regarded as an efficient tool for detecting promising regions in large solution spaces. However, they might not be well equipped for climbing these promising peaks. Rather than merely exploring the search space, an efficient algorithm also needs to perform some sort of local improvement or repair. The association *"repair algorithm and GA"* might provide a better balance between exploration and exploitation; the GA explores the search space and the repair algorithm performs a local search in the neighbourhood of the solutions provided by the GA.

This hybrid algorithm follows the same cycle as the previous steady-state GAs. However, once the crossover operator has generated a new solution, this solution is transferred to the repair algorithm. Once the repair algorithm has completed its task, the newly modified solution is finally inserted in the population.

The main objective of the repair algorithm is to improve the quality of the schedules. This can be achieved in two different ways:

1. Reduce the total distance travelled by the engineers.

2. Reduce the amount of work not done. This option provides the best scope for improvement. A reduction in work not done will generate a significant reduction of cost.

The repair algorithm, adopted in this study, works as follows: first, the VRP is viewed as a set of independent TSPs and a tour improvement heuristic [BRIN94] is used to rearrange the different tours. The main motivation here, is not a mere reduction of travel, rather the main concern is to rearrange the tours, so that each engineer spends a minimum time travelling and a maximum time working. Once

[5] Observe that a repair algorithm is already implemented in each direct crossover. The repair algorithm, presented in this section, is a more complex algorithm.

this first stage is completed, the repair algorithm attempts to insert unallocated jobs in the schedule where the tour improvement heuristic has created some empty slots.

In this section, Direct4-s:200, Dupli-1 and Dupli-2 are coupled with the repair algorithm. Table 6.15 presents the final costs of six algorithms: Direct4-s:200, Direct4-s:200 with repair, Dupli-1, Dupli-1 with repair, Dupli-2 and finally Dupli-2 with repair.

Table 6.16 gives the ranking of these algorithms over the different problems; this information is summarised in Figure 6.6 which gives the average, the minimum and the maximum scores for each algorithm over the entire set of problems. In this case, one algorithm can score a maximum of 5 points and a minimum of 0 points for a given problem.

Algorithm	average	min	max
Direct4-s:200	1.38	0	5
Direct4-s:200 + repair	2.63	0	5
Dupli-1	2.75	1	5
Dupli-1 + repair	2.66	0	5
Dupli-2	2.22	0	5
Dupli-2 + repair	3.33	0	5

Figure 6.6: Ranking of GAs with or without repair algorithm.

Overall, the addition of the repair algorithm appears to be beneficial. The average score of Direct4-s:200 jumps from 1.38 to 2.63 when the repair algorithm is added (and from 2.22 to 3.33 for Dupli-2). However, Dupli-1 does not seem to benefit from this association, its average score drops from 2.75 down to 2.66. These last results should be considered carefully as the difference between the two scores is minimal.

Before the addition of the repair algorithm, the ranking between the algorithms was

Dupli-1 > Dupli-2 > Direct4-s:200 (where the relation ">" is read as "generally achieves a better score than").

The repair algorithm modifies the previous ranking, which now becomes Dupli-2 with repair > Dupli-1 with repair > Direct4-s:200. This modification is due mainly to the fact that Dupli-1 does not take advantage of the repair algorithm.

Dupli-2 with repair has the best score overall and it performs well on 50-* problems. However, this algorithm finds relatively poor solutions for under-resourced problems.

With under-resourced and critical problems, i.e. all 30-* and some 40-* problems, there is no clear champion. Direct4-s:200 manages 3 pole-positions, Direct4-s:200 with repair also manages 3, Dupli- 1 manages 4, Dupli-1 with repair manages 6, Dupli-2 manages 1 and finally Dupli-2 with repair also manages 1 pole- position.

The repair algorithm is especially effective on the 50-* problems. Furthermore, Table 6.15 indicates that the repair algorithm is especially useful when solving problems with tight technological constraints. This result is quite unexpected. Indeed, technological constraints limit the possible rearrangements of the different tours and, thus a high level of technological constraint should reduce the scope for improvement.

Table 6.17 shows the amount of exploration performed by the different algorithms in 10 minutes. As expected, the addition of the repair algorithm drastically alters the computational behaviour of the different algorithms: the number of iterations performed in 10 minutes drops considerably when the repair algorithm is added. At each iteration, the repair algorithm rearranges all the tours of the current solutions and then tries to insert unallocated jobs in the schedule. These two operations require a significant amount of time, and this overhead means that the GA is performing fewer iterations. In other words, the hybrid algorithm is doing more exploitation and less exploration. As a direct consequence, a smaller number of

solutions are visited. However, it also means that each one of these iterations is likely to generate a better solution.

Finally, Table 6.18 shows for the 6 algorithms when the best solutions were found (in %) for the different problems. The principal observation is that, when coupled with a repair algorithm, a GA tends to find its best solution at a later stage. This is especially true for problems with tight technological constraints. This result suggests that these hybrid algorithms would also benefit from longer runs. Each iteration demands more computational effort and 10 minutes is apparently not a sufficient for the GA to perform sufficient exploration.

In general, the association GA-repair algorithm is fruitful and this is especially true for over-resourced problems. This hybrid algorithm has a different behaviour from a *"pure 100 %"* GA; it performs fewer iterations, i.e. less exploration and more exploitation. Results suggest that longer runs could be beneficial.

	Direct4-s 200	Direct4-s 200 repair	Dupli-1	Dupli-1 repair	Dupli-2	Dupli-2 repair
30-0-0-1	526785	553813	535779	615070	559222	609660
30-0-0-2	319818	321592	308984	393661	341434	364844
30-0-0-3	175831	173961	193840	251505	201057	231668
30-0-3-1	374021	348769	301907	312679	413671	377560
30-0-3-2	451450	373937	379346	361335	438829	424366
30-0-3-3	712334	712298	690690	685255	744747	706887
30-2-0-1	582873	584639	645932	683744	636926	696327
30-2-0-2	554164	550484	577598	595691	550580	601013
30-2-0-3	1380200	1396475	1387475	1419982	1387506	1430746
30-2-3-1	1123114	1124798	1088868	1097834	1126788	1106827
30-2-3-2	851285	806175	802645	788203	854868	851209
30-2-3-3	750542	718002	691023	669351	701913	716188
40-0-0-1	1249	1177	1240	1224	1150	1100
40-0-0-2	1256	1172	1199	1176	1123	1061
40-0-0-3	1215	1166	1212	1204	1135	1026
40-0-3-1	1450	1380	1416	1378	1402	1352
40-0-3-2	1394	1332	1329	1326	1397	1297
40-0-3-3	1512	1436	1458	1414	1424	1392
40-2-0-1	370864	352730	345682	378092	342030	370829
40-2-0-2	1750	1739	1824	1806	9033	1730
40-2-0-3	1717	1679	1739	1821	1832	1724
40-2-3-1	1960	2003	1845	1863	2042	1950
40-2-3-2	1838	1705	1691	1668	1839	1745
40-2-3-3	46982	64915	27135	23461	61440	46922
50-0-0-1	1276	1246	1289	1297	1139	1179
50-0-0-2	1252	1171	1252	1255	1174	1067
50-0-0-3	1210	1152	1182	1220	1115	1044
50-0-3-1	1513	1429	1447	1422	1423	1360
50-0-3-2	1519	1454	1420	1386	1443	1354
50-0-3-3	1477	1469	1443	1443	1442	1369
50-2-0-1	1628	1566	1589	1598	1527	1534
50-2-0-2	1487	1386	1479	1446	1352	1326
50-2-0-3	1498	1457	1454	1425	1436	1400
50-2-3-1	1771	1675	1700	1642	1725	1583
50-2-3-2	1965	1886	1836	1824	1912	1862
50-2-3-3	1882	1877	1789	1793	1851	1756

Table 6.15: Effect of repair algorithm - Average final cost.

	Direct4-s 200	Direct4-s 200 repair	Dupli-1	Dupli-1 repair	Dupli-2	Dupli-2 repair
30-0-0-1	5	3	4	0	2	1
30-0-0-2	4	3	5	0	2	1
30-0-0-3	4	5	3	0	2	1
30-0-3-1	2	3	5	4	0	1
30-0-3-2	0	4	3	5	1	2
30-0-3-3	1	2	4	5	0	3
30-2-0-1	5	4	2	1	3	0
30-2-0-2	3	5	2	1	4	0
30-2-0-3	5	2	4	1	3	0
30-2-3-1	2	1	5	4	0	3
30-2-3-2	1	3	4	5	0	2
30-2-3-3	0	1	4	5	3	2
40-0-0-1	0	3	1	2	4	5
40-0-0-2	0	3	1	2	4	5
40-0-0-3	0	3	1	2	4	5
40-0-3-1	0	3	1	4	2	5
40-0-3-2	1	2	3	4	0	5
40-0-3-3	0	2	1	4	3	5
40-2-0-1	1	3	4	0	5	2
40-2-0-2	3	4	1	2	0	5
40-2-0-3	4	5	2	1	0	3
40-2-3-1	2	1	5	4	0	3
40-2-3-2	1	3	4	5	0	2
40-2-3-3	2	0	4	5	1	3
50-0-0-1	2	3	1	0	5	4
50-0-0-2	1	4	2	0	3	5
50-0-0-3	1	3	2	0	4	5
50-0-3-1	0	2	1	4	3	5
50-0-3-2	0	1	3	4	2	5
50-0-3-3	0	1	2	3	4	5
50-2-0-1	0	3	2	1	5	4
50-2-0-2	0	3	1	2	4	5
50-2-0-3	0	1	2	4	3	5
50-2-3-1	0	3	2	4	1	5
50-2-3-2	0	2	4	5	1	3
50-2-3-3	0	1	4	3	2	5

Table 6.16: Effect of repair algorithm - Ranking of the algorithms.

	Direct4-s 200	Direct4-s 200 repair	Dupli-1	Dupli-1 repair	Dupli-2	Dupli-2 repair
30-0-0-1	8150	3170	4720	2020	4830	2050
30-0-0-2	11450	4040	5540	2280	6770	2550
30-0-0-3	15900	4830	5870	2530	9260	2810
30-0-3-1	37800	9990	32540	8110	37560	8900
30-0-3-2	38340	9770	34510	8230	35260	8750
30-0-3-3	27140	8760	24410	7280	25350	7560
30-2-0-1	6660	2860	3720	1930	4150	1940
30-2-0-2	6970	2970	3880	2030	4060	2090
30-2-0-3	4100	2010	3000	1560	2990	1560
30-2-3-1	18470	7080	17540	6210	17910	6640
30-2-3-2	20830	8120	19680	6830	21140	7300
30-2-3-3	22380	8920	21990	7350	23280	7890
40-0-0-1	27660	7770	9270	4790	72060	7000
40-0-0-2	25080	7590	7650	4490	60810	6740
40-0-0-3	26550	8040	8550	4640	65450	7260
40-0-3-1	89390	14280	45250	11830	150930	14630
40-0-3-2	85160	13780	42860	10940	153620	14570
40-0-3-3	83880	14260	49660	11900	153150	15100
40-2-0-1	11950	4480	6290	3020	8980	3020
40-2-0-2	54100	9890	10760	4680	50350	6530
40-2-0-3	42580	10280	8830	4590	63510	7630
40-2-3-1	99340	17590	62880	13850	151400	17240
40-2-3-2	92290	16530	55920	12460	149370	16340
40-2-3-3	80080	15340	72390	12280	83900	14230
50-0-0-1	30140	8160	10140	4810	60620	7140
50-0-0-2	25650	7740	8040	4650	64610	6530
50-0-0-3	25570	6940	9900	4790	57250	7310
50-0-3-1	76650	13770	44190	12020	132980	14510
50-0-3-2	81400	13730	45200	11510	132790	14510
50-0-3-3	76610	13860	44030	11810	134910	14800
50-2-0-1	41920	10290	14120	6420	71640	10990
50-2-0-2	31950	9800	13300	6100	74980	10270
50-2-0-3	30160	9420	14100	6710	84770	10660
50-2-3-1	104020	16950	55640	13410	140940	16620
50-2-3-2	99450	16230	48810	12710	126290	16210
50-2-3-3	93960	17190	56280	14010	137980	17000

Table 6.17: Effect of repair algorithm - Amount of iteration in 10 minutes.

	Direct4-s 200	Direct4-s 200 repair	Dupli-1	Dupli-1 repair	Dupli-2	Dupli-2 repair
30-0-0-1	62	94	96	78	99	93
30-0-0-2	54	87	91	82	99	97
30-0-0-3	76	94	97	88	99	95
30-0-3-1	61	84	64	97	99	99
30-0-3-2	40	56	67	93	99	99
30-0-3-3	41	57	62	95	99	99
30-2-0-1	89	92	93	92	97	89
30-2-0-2	80	95	94	92	98	96
30-2-0-3	82	94	92	95	98	77
30-2-3-1	52	65	73	91	83	99
30-2-3-2	64	61	62	93	99	99
30-2-3-3	63	69	90	86	99	99
40-0-0-1	42	87	94	91	99	99
40-0-0-2	64	86	97	93	99	96
40-0-0-3	59	71	90	90	99	99
40-0-3-1	45	60	73	92	99	99
40-0-3-2	34	72	56	91	99	99
40-0-3-3	34	73	67	91	99	99
40-2-0-1	73	94	96	96	99	96
40-2-0-2	41	77	88	95	99	99
40-2-0-3	48	70	96	97	99	99
40-2-3-1	27	67	76	83	99	99
40-2-3-2	24	59	52	90	99	99
40-2-3-3	18	43	64	90	99	99
50-0-0-1	58	75	81	93	99	99
50-0-0-2	53	84	94	91	99	99
50-0-0-3	40	76	93	95	99	99
50-0-3-1	47	74	75	86	99	99
50-0-3-2	56	56	68	94	99	99
50-0-3-3	35	71	74	85	99	99
50-2-0-1	48	80	87	91	99	99
50-2-0-2	44	65	94	93	99	99
50-2-0-3	39	85	84	88	99	99
50-2-3-1	46	75	55	80	99	99
50-2-3-2	47	65	59	89	99	99
50-2-3-3	47	73	65	86	99	99

Table 6.18: Effect of repair algorithm - When the best solution was found (in % of 10 minutes).

6.2 The 10 minute Beauty Contest.

This section examines and compares the performances of five algorithms; namely, genetic algorithms (GAs), hill-climbing (HC), random search (RS), simulated annealing (SA) and tabu search (TS) over the entire range of problems. In all trials the computational resource was limited to 10 SPARCstation IPC minutes per algorithm per problem. Hence, the term *"10 minute beauty contest"*. The experiments reported in this section required over 150 hours of computational effort.

The five algorithms were implemented with the following parameters:

- RS used, again, as the reference algorithm. The results for this algorithm are the same as the results presented in the previous sections.
- Dupli-1 was chosen as the *"GA candidate"*. Among all the GAs, Dupli-2 with repair had the best score overall; however, it found relatively poor solutions for the 30-*-*-* problems. Hence, Dupli-1 was chosen primarily because it finds slightly better solutions than the other GAs on under-resourced and critical problems. For information, this GA used the knowledge-based Crossover #4, the steady-state model and the exponential selection function; it had no mutation operator and operated with a population of 400 chromosomes. It accepted duplicates but only during the initial building of the population. Thereafter, duplicates were rejected. Finally, Dupli-1 was not coupled with the repair algorithm because of the length of the runs and to allow enough exploration.
- TS starts its search with an initial random solution created with the schedule-builder (this is also true for HC and SA). Then, the algorithm iterates. The move operator, described in the next paragraph, used by TS is common to the three neighbourhood search techniques. TS explores an entire neighbourhood at each iteration. Here, the size of a neighbourhood is $n + 1$ solutions

where there are n engineers and one conflict-list. In order to create the nth neighbour, TS takes the current solution and randomly selects a job J_i from the nth engineer's tour. Then, TS repeatedly selects engineers randomly until J_i fits in the selected engineer's tour. Once the $n+1$ neighbours have been generated, the best solution in the neighbourhood (S_{best}) becomes the current solution. However, before being accepted, S_{best} must satisfy the restriction criteria which is to forbid moving jobs which have already been moved within the tabu tenure; here, 125 moves. This obligation is waived when S_{best} meets the aspiration criteria, i.e. the cost of S_{best} has to be less than the cost of the best solution found so far. Further information concerning the tuning of TS is available in [PROS94] and [BRIN94].

- In a similar manner to TS, HC generates $n+1$ neighbours at each iteration around the current solution S. Once this has been completed, the best solution S_{best} in the neighbourhood is compared with S. The difference in cost between S and S_{best} is calculated. If S_{best} has a lower cost, the move is accepted and S_{best} becomes S, otherwise S remains unchanged. This cycle is then repeated until the time limit is reached.

- SA attempts a given number of moves at a temperature $Temp$, then reduces the temperature by a given amount. That is, $Temp = Temp \times R$, where R is the cooling rate. In the experiments performed, the initial temperature was set to $Temp = 10.0$, the cooling rate $R = 0.992$ and the number of successful moves at a given temperature was 500. These values provided a very smooth cooling schedule. The search process is terminated after 10 minutes. SA uses the following rules to accept or reject a neighbourhood solution S':

  - if $cost(S') \leq cost(S)$ then S' is accepted as the new current solution.
  - if $cost(S') > cost(S)$ then the case is treated in a probabilistic manner.

In this case, the probability of acceptance is given by:

$$P(\Delta cost) = e^{-Temp/\Delta} \leq random(1.0)$$

where $Temp$ is the control parameter temperature, $\Delta cost = cost(S') - cost(S)$, and random (1.0) generates a random number in the range 0.0 to 1.0, drawn from the uniform distribution. If $P(\Delta cost)$ is greater than the random number then S' is accepted as the new current solution, otherwise it is rejected and the original solution S remains the current solution.

The move used by TS, HC, and SA works as follows. A Job J_i is randomly selected. This job is then removed from the tour of the relevant engineer E_x and an attempt is made to insert the job into the tour of some other randomly selected engineer E_y, where E_y is technically able to do J_i. Note that E_y may be the conflict-list, i.e. E_y may be the virtual engineer, so long as J_i is not compulsory. The move may:

1. increase the number of jobs being done if E_x is the conflict-list.
2. modify, i.e. increase or decrease, the total travel by reducing the tour length of E_x and increasing the tour length of E_y.
3. decrease the amount of work done if E_y is the conflict list (the virtual engineer).

Moreover, if E_y is the conflict list, i.e. if the algorithm is moving a job J_i from E_x into the conflict-list, the algorithm immediately attempts to move a job J_j from the conflict-list back to engineer E_x. Moving J_i into the conflict-list will always generate a large increase in cost, and moving J_j back in the schedule will always generate a large reduction in cost. If job J_i has a longer duration than job J_j, the moves will most likely be rejected. Otherwise, the moves will (again, most likely) be accepted.

Table 6.19 presents the performances of the 5 algorithms over the entire range of problems and Table 6.20 presents the ranking of the algorithms. Figure 6.7 provides the average score, the minimum and the maximum for the five algorithms. In this case, each algorithm can score a maximum of 4 points and a minimum of 0 points.

Algorithm	average	min	max
GA	1.58	1	4
HC	1.91	0	3
RS	0.03	0	1
SA	3.88	2	4
TS	2.58	2	4

Figure 6.7: Ranking of the search techniques for 10 minute runs.

Figure 6.7 shows that SA, with an impressive average score of 3.88, dominates its opponents; SA is in fact only beaten on two occasions: on the 30-0-3-1 problem by GA and on the 30-2-3-3 problem by TS. GA performs relatively well on under-resourced problems; however, it achieves some rather poor results when applied to over-resourced problems. Typically, GA is only able to beat RS on these problems. This last remark explains the low score (1.58) of GA. In other words, GA is quite capable of packing jobs into an under-resourced schedule; however, it cannot reduce travel as well as the three neighbourhood search techniques. This means that when the reduction of travel becomes the only scope for improvement, i.e. on over-resourced problem, GA cannot compete with SA, TS, and HC.

The poor performance of GA on over-resourced problems has a number of causes. First, the GA adopted for this comparative study, Dupli-1, uses Crossover #4. This knowledge-based crossover is not recommended for the solution of over-resourced problems as it might misguide the search (see Subsection 6.1.1). Clearly, a GA using a random-based crossover (such as Crossover #3) would have achieved better results for these particular problems. Secondly, in an over-resourced problem, a significant proportion of the engineers are idle, and have empty tours. In a direct

representation when performing crossover, the GA may inherit many empty tours and this is of no real value, i.e. each time this happens, the GA is losing one opportunity of mixing good tours. Finally, once the inheritance phase is completed, the GA tries to bind each unallocated job to any possible engineer. It might be a better idea to allocate jobs first to existing tours and then if any job remains unallocated, consider idle engineers.

While HC is dominated by SA, GA and TS on under-resourced problems, it performed relatively well on over-resourced problems. However, the algorithm finds it difficult to maintain its level of performance when the level of constraints (both temporal and technological) increases, and on the 40-0-3-*, 40-2-0-*, 40-2-3-*, 50-2-3-*, typically HC only manages to beat RS and GA. TS has the second highest score (2.58) behind SA. Like the other neighbourhood search techniques, TS performs well on over-resourced problems and also performs fairly well on under-resourced problems, especially when the level of constraint is significant.

Table 6.21 shows the amount of exploration performed by the five algorithms. The GA column shows the number of crossovers and the RS column shows the number of schedules created in 10 minutes. The HC and TS column give the number of neighbourhoods examined; where a neighbourhood is the number of engineers in the problem plus one (for the conflict list). Therefore, the 50-* problems have a larger neighbourhood than the 30-* problems. The SA column gives the number of temperature reductions. There must be 500 successful moves before the temperature is reduced. Comparisons should be made downwards through the table not across rows.

The five algorithms fall into 3 categories: blind, neighbourhood and knowledge intensive techniques. RS is a blind technique, which does not use the history of the search in any way [6] and creates and evaluates a new solution from scratch at each

[6]The exception being that RS compares the current solution with the best solution found so far.

iteration. These two operations are extremely expensive in terms of computational effort and hence, RS can only visit a limited number of points in the search space. HC, SA, and TS are neighbourhood techniques. When generating new solutions these techniques only need to generate a simple move, i.e. the deletion of a job from an engineer's tour and the addition of that same job to the tour of another engineer, and evaluate the effect of that simple move. These two operations are relatively inexpensive and HC, SA, and TS can explore a large number of points in the search space (these points might be quite similar to each other).

Compared to the four other techniques, GA is a knowledge intensive technique which attempts to exploit the information, i.e. the genetic material, present in a pool of candidate-solutions. The algorithm combines two solutions into a child and evaluates that child. These two operations are a few orders of magnitude more expensive than the operations performed by HC, SA, and TS. Therefore, GA performs far fewer iterations, i.e. less exploration but more exploitation, than the three neighbourhood techniques.

When creating the child, GA does not have to re-schedule the entire solution, parts of both parents are inherited without any alteration. Therefore, GA performs more iterations than RS. Moreover, more exploration takes places over problems with high specialisation. This phenomenon is observed with the five algorithms. Also, when moving from under-resourced to over-resourced, the five algorithms performed significantly more iterations.

Table 6.22 shows when the different algorithms find their best solution (in %) over the entire set of problems. When moving from under-resourced to over-resourced problems, the different algorithms find their best solution earlier in the 10 minutes; they also find their best solution earlier when applied to problems with high specialisation. Clearly, on these two classes of problems, the algorithms are performing more iterations and reach their best solutions more rapidly. In general, GA finds

its best solution at a later stage than the other algorithms [7]. This again suggests that GA could benefit from longer runs.

In conclusion, given 10 minutes, SA is the best technique overall. In an environment where the algorithm is given only a limited amount of time, SA is a robust technique which performs well both on over-resourced and under-resourced problems. GA works fairly well on under-resourced problems; however, it is outperformed by the neighbourhood search techniques (SA, HC, and TS) on over-resourced problems. This is due mainly to the fact that GA does not *reduce travel* as well as SA, HC, and TS.

[7] This might be partly due to the fact that GA has to create an initial population before starting its search

	GA	HC	RS	SA	TS
30-0-0-1	535779	580803	975696	440246	559202
30-0-0-2	308984	399073	775956	258495	348610
30-0-0-3	193840	244296	617570	107333	206448
30-0-3-1	301907	449678	763409	337981	337968
30-0-3-2	379346	498310	815645	350548	379354
30-0-3-3	690690	778948	1092715	638426	676248
30-2-0-1	645932	604391	909188	505374	550357
30-2-0-2	577598	561250	772461	465438	492380
30-2-0-3	1387475	1399526	1499448	1354369	1368904
30-2-3-1	1088868	1225485	1364732	1052343	1077584
30-2-3-2	802645	924950	1154086	770189	793561
30-2-3-3	691023	760957	909303	697780	672561
40-0-0-1	1240	920	2246	768	1157
40-0-0-2	1199	897	2231	785	1176
40-0-0-3	1212	934	2186	761	1079
40-0-3-1	1416	1310	2315	963	1172
40-0-3-2	1329	1166	2200	889	1078
40-0-3-3	1458	1305	2382	998	1199
40-2-0-1	345682	335799	445075	254771	258469
40-2-0-2	1824	11894	54407	898	1040
40-2-0-3	1739	1109	56334	925	1072
40-2-3-1	1845	55402	22329	1155	1218
40-2-3-2	1691	1320	2279	1021	1056
40-2-3-3	27135	157962	191368	13802	46245
50-0-0-1	1289	990	2313	818	1248
50-0-0-2	1252	900	2235	817	1170
50-0-0-3	1182	825	2234	770	1097
50-0-3-1	1447	1176	2276	941	1216
50-0-3-2	1420	1218	2365	974	1199
50-0-3-3	1443	1166	2325	972	1172
50-2-0-1	1589	1067	2184	897	1080
50-2-0-2	1479	875	2107	718	971
50-2-0-3	1454	967	2119	826	1010
50-2-3-1	1700	1244	2184	1000	1079
50-2-3-2	1836	1374	2354	1136	1189
50-2-3-3	1789	1304	2385	1109	1154

Table 6.19: 10 minute runs - Average final cost.

	GA	HC	RS	SA	TS
30-0-0-1	3	1	0	4	2
30-0-0-2	3	1	0	4	2
30-0-0-3	3	1	0	4	2
30-0-3-1	4	1	0	2	3
30-0-3-2	3	1	0	4	2
30-0-3-3	2	1	0	4	3
30-2-0-1	1	2	0	4	3
30-2-0-2	1	2	0	4	3
30-2-0-3	2	1	0	4	3
30-2-3-1	2	1	0	4	3
30-2-3-2	2	1	0	4	3
30-2-3-3	3	1	0	2	4
40-0-0-1	1	3	0	4	2
40-0-0-2	1	3	0	4	2
40-0-0-3	1	3	0	4	2
40-0-3-1	1	2	0	4	3
40-0-3-2	1	2	0	4	3
40-0-3-3	1	2	0	4	3
40-2-0-1	1	2	0	4	3
40-2-0-2	2	1	0	4	3
40-2-0-3	1	2	0	4	3
40-2-3-1	2	0	1	4	3
40-2-3-2	1	2	0	4	3
40-2-3-3	3	1	0	4	2
50-0-0-1	1	3	0	4	2
50-0-0-2	1	3	0	4	2
50-0-0-3	1	3	0	4	2
50-0-3-1	1	3	0	4	2
50-0-3-2	1	2	0	4	3
50-0-3-3	1	3	0	4	2
50-2-0-1	1	3	0	4	2
50-2-0-2	1	3	0	4	2
50-2-0-3	1	3	0	4	2
50-2-3-1	1	2	0	4	3
50-2-3-2	1	2	0	4	3
50-2-3-3	1	2	0	4	3

Table 6.20: 10 minute runs - Ranking of the search techniques.

	GA	HC	RS	SA	TS
30-0-0-1	4720	3240	1400	190	3140
30-0-0-2	5540	3280	1400	190	3100
30-0-0-3	5870	3200	1400	190	3100
30-0-3-1	32540	10840	4400	660	10400
30-0-3-2	34510	10520	4400	652	10120
30-0-3-3	24410	11040	4600	648	10360
30-2-0-1	3720	3560	1400	190	3200
30-2-0-2	3880	4040	1420	230	5120
30-2-0-3	3000	7440	1500	458	6760
30-2-3-1	17540	12600	4680	758	12360
30-2-3-2	19680	11240	4800	626	10240
30-2-3-3	21990	14960	5200	930	14600
40-0-0-1	9270	17840	1400	1228	20180
40-0-0-2	7650	19120	1400	1272	21260
40-0-0-3	8550	17320	1400	1200	19600
40-0-3-1	45250	22560	4500	1548	22040
40-0-3-2	42860	22080	4500	1506	21120
40-0-3-3	49660	23240	4700	1544	21540
40-2-0-1	6290	11560	1500	1162	10720
40-2-0-2	10760	11640	1500	1240	12940
40-2-0-3	8830	12600	1500	1186	13060
40-2-3-1	62880	18600	4600	1592	18340
40-2-3-2	55920	19000	4600	1556	17920
40-2-3-3	72390	16320	4200	1386	15220
50-0-0-1	10140	23200	1400	1586	22540
50-0-0-2	8040	24400	1400	1588	22920
50-0-0-3	9900	25640	1400	1610	23700
50-0-3-1	44190	28040	4200	1842	23940
50-0-3-2	45200	27120	4300	1858	23300
50-0-3-3	44030	26240	4400	1914	23100
50-2-0-1	14120	18720	1500	1758	16000
50-2-0-2	13300	24560	1400	1802	19960
50-2-0-3	14100	20720	1500	1748	17320
50-2-3-1	55640	22520	4500	2038	19640
50-2-3-2	48810	21040	4000	2006	18700
50-2-3-3	56280	22600	4800	2050	20240

Table 6.21: 10 minute runs - Amount of exploration in 10 minutes.

	GA	HC	RS	SA	TS
30-0-0-1	96	85	-	95	86
30-0-0-2	91	72	-	93	83
30-0-0-3	97	72	-	91	70
30-0-3-1	64	21	-	46	58
30-0-3-2	67	7	-	50	65
30-0-3-3	62	21	-	52	81
30-2-0-1	93	51	-	72	74
30-2-0-2	94	20	-	100	81
30-2-0-3	92	53	-	85	69
30-2-3-1	73	13	-	46	81
30-2-3-2	62	6	-	45	60
30-2-3-3	90	7	-	38	64
40-0-0-1	94	41	-	61	15
40-0-0-2	97	65	-	50	24
40-0-0-3	90	73	-	47	39
40-0-3-1	73	23	-	35	38
40-0-3-2	56	50	-	31	46
40-0-3-3	67	70	-	28	54
40-2-0-1	96	94	-	47	48
40-2-0-2	88	15	-	41	47
40-2-0-3	96	27	-	51	54
40-2-3-1	76	23	-	26	70
40-2-3-2	52	36	-	28	66
40-2-3-3	64	16	-	31	86
50-0-0-1	81	59	-	37	15
50-0-0-2	94	63	-	43	22
50-0-0-3	93	63	-	70	20
50-0-3-1	75	49	-	29	24
50-0-3-2	68	77	-	26	33
50-0-3-3	74	74	-	27	36
50-2-0-1	87	19	-	30	24
50-2-0-2	94	24	-	43	17
50-2-0-3	84	16	-	38	16
50-2-3-1	55	27	-	22	46
50-2-3-2	59	31	-	26	51
50-2-3-3	65	45	-	21	41

Table 6.22: 10 minute runs - When the best solution is found (in % of 10 minutes).

6.3 CPU Intensive Experiments

This section analyses the performance of GA, HC, RS, SA, and TS when these algorithms are given significantly more time to explore the search space. The main motivation here is to check if the algorithms can benefit from these longer runs. Furthermore, these intensive runs provide the opportunity to verify the quality of the solutions provided by the short runs, i.e. is the difference between the solutions provided by the short runs and the intensive runs significant? Is the extra computational effort worthwhile?

RS and TS remain unchanged for these runs; their parameters keep the same values as in Section 6.2. The *GA candidate* is still Dupli-1 but it has a larger population of 2000 chromosomes rather than 400. For SA, the number of successful moves at a given temperature is set to 12000 moves instead of 500.

These intensive runs were performed on a DEC Alpha 3000/300 and each algorithm was given one hour on each problem. As previously, the algorithm was applied 5 times to each problem. Hence, to apply a single algorithm to all 36 problems took approximately 180 hours. These intensive runs were performed with 5 algorithms and took approximately 900 hours which is equivalent to 3600 hours on a SPARC-station IPC.

Algorithm	average	min	max	average - 10 min
GA	2.27	1	4	1.58
HC	1.55	0	3	1.91
RS	0.30	0	1	0.03
SA	3.6	2	4	3.88
TS	2.25	1	4	2.58

Figure 6.8: Ranking of the search techniques for 4 hour runs.

Table 6.23 presents the performances of the 5 algorithms over the entire range of