INTERACTIVE GAME
SCHEDULING WITH GENETIC
ALGORITHMS

By

Jason Leonard, B.Eng. (Hon)

A minor thesis submitted in partial
fulfillment of the requirements for
the degree of

Master Of Applied Science in
Information Technology

Department of Computer Science

Royal Melbourne Institute of
Technology University

1998
DECLARATION

I certify that all work on this thesis was carried out between February 1997 and June 1998 and it has not been submitted for any academic award at any other college, institute or university.

The work presented was carried out under the supervision of Dr. Vic Ciesielski, who also proposed the topic, liaised with the Australian National Basketball League, and reviewed progress.

All other work in the thesis is my own except where acknowledged in the text.

Signed,

Jason Matthew Leonard
June 1998.
ABSTRACT

INTERACTIVE GAME SCHEDULING WITH GENETIC ALGORITHMS

By Jason Leonard

Supervisor: Dr. Vic Ciesielski
Department of Computer Science
RMIT University

The potential of genetic algorithms in finding good solutions to the problem of scheduling games within round robin national sporting competitions is explored. An emphasis has been placed on an iterative scheduling paradigm, so that the human process of gradual refinement is supported. In particular, it is noted that ‘optimal’ solutions are unnecessary, since the weightings on the fixture requirements change as the scheduling process unfolds. Instead, good results that are obtained in a short time frame are desired.

The algorithm presented is designed to reduce the search space of the genetic algorithm by delegating certain aspects of fixture construction to a deterministic system. This permits usage of a short genetic encoding, resulting in faster convergence times and so allows interactive fixture development.

A software implementation of the algorithm allowed testing of variants of the genetic algorithm paradigm, and competitions of varying size are tested. Constraints from the Australian National Basketball League (NBL) were used to ground this testing in reality. It was found that a steady state genetic algorithm with a small population size is suitable for fixture generation for competitions with up to 18 teams. The NBL expects to test the software in upcoming seasons.
# Table of Contents

TABLE OF CONTENTS........................................................................................................................IV

LIST OF FIGURES ..............................................................................................................................VIII

LIST OF TABLES ...............................................................................................................................VIII

ACKNOWLEDGMENTS.......................................................................................................................IX

GLOSSARY..............................................................................................................................................X

CHAPTER 1: INTRODUCTION............................................................................................................1

CHAPTER 2: LITERATURE SURVEY .................................................................................................4

2.1 INTRODUCTION ..............................................................................................................................4

2.2 GENETIC ALGORITHMS................................................................................................................4

2.2.1 Simple Genetic Algorithms (SGA)...............................................................................................5

2.2.2 Maintaining Diversity..................................................................................................................6

2.2.3 Stopping a Genetic Algorithm....................................................................................................7

2.2.4 Replacement Methodologies....................................................................................................8

2.2.5 Advanced Selection Methodologies........................................................................................9

2.2.6 Crossover and Mutation..........................................................................................................9

2.2.7 Hybrid Genetic Algorithms...................................................................................................11

2.2.8 Deterministic Genetic Algorithms..........................................................................................11

2.3 SCHEDULING ALGORITHMS....................................................................................................13

2.3.1 Round Robin Sports Scheduling.............................................................................................13

2.3.2 The School Timetabling Problem............................................................................................15

2.3.3 Job Shop Scheduling..............................................................................................................19

2.3.4 Airline Crew Scheduling.........................................................................................................20

2.3.5 Scheduling with Graphs..........................................................................................................20

2.3.6 Interactive Scheduling...........................................................................................................21

2.4 THE COMBINATORICS OF ROUND ROBIN TOURNAMENTS ............................................22

2.4.1 NP-Completeness.....................................................................................................................22

2.4.2 Genetic Algorithms & NP-Complete Problems......................................................................23
2.4.3 Latin Squares .............................................................. 24
2.5 SOFTWARE PACKAGES FOR SPORTS SCHEDULING .................................................. 28

CHAPTER 3: THE DOMAIN ......................................................... 29
3.1 INTRODUCTION .............................................................. 29
3.2 PROBLEM SPECIFICATION .............................................. 29
3.3 CONSTRAINTS SPECIFIED BY THE NBL ........................................... 29
3.4 FURTHER ASSUMPTIONS AND CONSTRAINTS .............................. 31
3.5 DISCUSSION OF CONSTRAINTS ........................................... 31
3.5.1 Other Basketball Competitions ........................................ 31
3.5.2 Appropriateness ........................................................... 31
3.6 PROBLEM ANALYSIS ....................................................... 32

CHAPTER 4: THE ALGORITHM ................................................... 33
4.1 INTRODUCTION .............................................................. 33
4.2 THE ENCODING ............................................................... 33
4.2.1 A Naïve Encoding ......................................................... 33
4.2.2 A Short Encoding ........................................................ 35
4.2.3 Discussion of Short Encoding .......................................... 36
4.3 SCHEDULING GAMES ..................................................... 37
4.3.1 The Game Generation Algorithm .................................. 37
4.3.2 Choosing the Home Team ............................................. 38
4.3.3 Specifying the Game Date ............................................. 39
4.4 EXAMPLE ................................................................. 40
4.4.1 Who plays Whom? ........................................................ 40
4.4.2 Where do they Play? .................................................... 41
4.4.3 When are the Games? .................................................. 42
4.5 EVALUATING FIXTURES .................................................. 42
4.5.1 Introduction ............................................................... 42
4.5.2 Configuration ............................................................ 43
4.5.3 The Objective Function ................................................. 44
4.6 MAKING THE SCHEDULING INTERACTIVE ......................... 46

CHAPTER 5: EXPERIMENTAL RESULTS .................................... 47
5.1 INTRODUCTION .............................................................. 47
5.2 GENETIC ALGORITHM PARAMETERS ...................................... 48
List of Figures

Number Page
FIGURE 1: FIRST CODING 34
FIGURE 2: THE OBJECTIVE FUNCTION 45
FIGURE 3: FITNESS OF BEST FIXTURE VS. EXECUTION TIME FOR SS GA 51
FIGURE 4: FITNESS OF BEST FIXTURE VS. EXECUTION TIME FOR INCREMENTAL GA 52
FIGURE 5: FITNESS OF BEST FIXTURE VS. EXECUTION TIME FOR SIMPLE GA 53
FIGURE 6: MEAN EXECUTION TIME FOR DIFFERENT SIZE COMPETITIONS 58
FIGURE 7: MEAN EXECUTION TIME VS. NUMBER OF GAMES 58
FIGURE 8: GA OPERATION 78

List of Tables

Number Page
TABLE 1: LATIN SQUARE WITH UNIQUE DIAGONAL 24
TABLE 2: ROUND ROBIN TOURNAMENT, CONSTRUCTED FROM LATIN SQUARE 26
TABLE 3: RESULTS FROM STEADY STATE GA 51
TABLE 4: RESULTS FROM INCREMENTAL GA 52
TABLE 5: RESULTS FROM SIMPLE GA 53
TABLE 6: BREAKDOWN OF HIGHEST SCORING FIXTURE 55
TABLE 7: BREAKDOWN OF SCORE FROM ALGORITHM WITHOUT DATE AVAILABLE PENALTY 56
TABLE 8: RESULTS FOR 18 TEAM DOUBLE ROUND ROBIN COMPETITION WITH POPULATION SIZES OF 50 AND 500. 59
TABLE 9: SAMPLE OF NBL FIXTURE FORMAT (PORTION ONLY) 61
Acknowledgments

The software for this work used the GAlib genetic algorithm package (version 2.3.2), written by Matthew Wall at the Massachusetts Institute of Technology.

The author wishes to thank Bill Palmer of the NBL, for providing the opportunity and information to take on this project; my supervisor, Vic Ciesielski, for reviewing my ideas and drafts of this thesis; and my family, Louise & Patrick, for remembering who I am.
## Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch Scheduling</td>
<td>A scheduling system that does not allow a user to interrupt its execution to modify constraints or other scheduling parameters.</td>
</tr>
<tr>
<td>Break</td>
<td>Two consecutive games that are both played either at home or away.</td>
</tr>
<tr>
<td>Bye</td>
<td>A team has a ‘bye’ when it is not scheduled to play in a given weekend.</td>
</tr>
<tr>
<td>Chromosome</td>
<td>Also called a ‘string’, it contains genetic material in the form of genes. Also see Genome.</td>
</tr>
<tr>
<td>Crossover</td>
<td>The creation of new genomes using the genomes of two or more parents.</td>
</tr>
<tr>
<td>Double Round Robin</td>
<td>A tournament in which every team plays all other teams exactly twice. See also Round Robin.</td>
</tr>
<tr>
<td>Epistasis</td>
<td>The extent to which the contribution to fitness of one gene depends on the values of other genes. Problems with little or no epistasis are trivial to solve (hillclimbing is sufficient). Highly epistatic problems are difficult to solve because it is difficult to form building blocks.</td>
</tr>
<tr>
<td>Feasible Solution</td>
<td>A solution to the problem that could be implemented in the ‘real world’.</td>
</tr>
<tr>
<td>Fixture</td>
<td>A list of games for a season in a sporting competition, with the dates, venues and names of the teams playing.</td>
</tr>
<tr>
<td>GA</td>
<td>Genetic Algorithm; a group of optimisation techniques that are loosely based on Darwin’s theory of evolution.</td>
</tr>
<tr>
<td>Genome</td>
<td>A genome contains all the chromosomes in an individual. In this thesis, only a single chromosome is used so the term is used interchangeably.</td>
</tr>
<tr>
<td>Hard Constraint</td>
<td>A constraint that must be satisfied for the result to be considered feasible.</td>
</tr>
<tr>
<td>HCM</td>
<td>Heuristic Combination Method; genetic values are heuristics, and the objective is to find the best sequence of heuristics.</td>
</tr>
<tr>
<td>Home game</td>
<td>A game played at a team’s home venue. Each team in the NBL has a home venue, though some teams share a venue.</td>
</tr>
<tr>
<td>Interactive Scheduling</td>
<td>The opposite of batch scheduling. Discussed further in the Introduction.</td>
</tr>
<tr>
<td>Lethal</td>
<td>An individual in a population in a genetic algorithm with a fitness far lower than the mean of the population.</td>
</tr>
<tr>
<td>JSSP</td>
<td>The job shop scheduling problem. Usually this is set in a manufacturing environment, and the objective is to produce a given number of components on different machines at the lowest cost and/or the fastest time.</td>
</tr>
<tr>
<td>Mutation</td>
<td>Random modification of genetic material within a population.</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NBL</td>
<td>National Basketball League. This is Australia’s premier basketball competition.</td>
</tr>
<tr>
<td>OR</td>
<td>Operations Research - a branch of applied mathematics that concentrates on providing formalised solutions to ‘real world’ problems.</td>
</tr>
<tr>
<td>Resampling</td>
<td>Two individuals that have identical genetic material are sampling the same point in the search space - resampling is occurring.</td>
</tr>
<tr>
<td>Road Trip</td>
<td>When a team plays successive away games, they are said to be ‘on the road’, or ‘on a road trip’. More specifically, this thesis refers to a road trip as a pair of games played against two teams that are Road Trip Partners.</td>
</tr>
<tr>
<td>Road Trip Partner (RTP)</td>
<td>A RTP is a team that is linked to another team in a fixture, so that they often play the same opponent at home in successive rounds.</td>
</tr>
<tr>
<td>Round</td>
<td>A group of games where each team play every other team. (Exception: one team may have a 'bye' if there is an odd number of teams).</td>
</tr>
<tr>
<td>Round Robin</td>
<td>A tournament system where each team must play every other team at least once. Games are often arranged into rounds.</td>
</tr>
<tr>
<td>Schema</td>
<td>A schema describes a chromosome in terms of the alphabet and an additional ‘don’t care’ symbol (usually ‘*’). Also known as a ‘similarity template’. [See Goldberg, p19]</td>
</tr>
<tr>
<td>SGA</td>
<td>Simple Genetic Algorithm, as specified in [Goldberg].</td>
</tr>
<tr>
<td>Secondary Venue</td>
<td>A backup stadium used when the primary venue is unavailable. The stadium is used as a backup only because of seating capacity, poor amenities, or even high cost to the league or association.</td>
</tr>
<tr>
<td>Selection</td>
<td>The selection of an individual within a genetic algorithm for reproduction.</td>
</tr>
<tr>
<td>Soft Constraint</td>
<td>A constraint that should be satisfied if possible. The degree to which soft constraints are satisfied is often used to differentiate one solution from another.</td>
</tr>
<tr>
<td>TSP</td>
<td>The Travelling Salesperson Problem - try to find the shortest distance to travel if a salesperson must visit each town in an itinerary exactly once.</td>
</tr>
</tbody>
</table>
Chapter 1: Introduction

Scheduling in many different areas falls into the category of ‘NP-complete’ problems; i.e. current algorithms require exponential time to reach a solution [see section 2.4, page 22]. These problems demand innovative solutions if they are to be solved within a reasonable amount of time. Further, scheduling problems come in many different forms, and so many human schedulers use various (manual) heuristic methods, learned only with hard won experience. The resulting schedules are often far from optimal, and yet have taken many hours to produce.

Scheduling, on the surface, seems to be a prime candidate for automation by a computer; since one would imagine a computer could apply the heuristics used by human schedulers far more quickly. However, as is often the case with ‘rules of thumb’, the people who use them often find it difficult to explain how and when they apply the rules, and even what the rules are. Nevertheless, commercial software packages [see section 2.5, page 28] are available for local sporting competitions.

The constraints involved in scheduling a televised national sporting competition are usually in addition to those of a local competition, and therefore add extra complexity. The administration of Australia’s National Basketball League, for example, must try to maximize gate takings by not scheduling too many consecutive home games for any team, and must also try to maximize television ratings to enhance sponsorship deals.

This thesis explores whether genetic algorithms can assist in round robin tournament scheduling tasks by applying them to the specific problem of scheduling the NBL. The research will specifically try to find a genetic algorithm that makes automatic iterative scheduling practical for modern but
relatively low cost computing equipment. This may be achieved by using an efficient encoding, or by using an advanced algorithm.

The title of this thesis uses the word 'interactive'. Unfortunately, this term is ambiguous as the literature and commercial scheduling packages use it in different ways; for example:

1. The user can 'interact' with a graphical user interface, usually when the schedule can be displayed as a Gantt chart. The user can then visually 'blot out' unavailable time slots and fix activities in time. [For example software, see Planning Windows at http://www.pst.stsci.edu/spss, or Winterface at http://www.autom.dist.unige.it/AUTOM_LAB/software.html]. Note that a Gantt chart approach cannot be used with a sporting fixture; however, similar results could be obtained with a tabular form.

2. Batch style scheduling software can create (possibly multiple) schedules, and then allow the user to modify the constraints and reschedule. [See http://www.ems-ltd.co.uk/preactor.htm for the Preactor system, which operates in this manner]. A special case of this is where the scheduling is iteratively generated, with the user fixing portions of the schedule as he or she becomes happy with it. For example, a sporting league fixture may suitable to the user in all bar a handful of games. This may be because there are constraints that cannot be predicted in advance, such as television coverage or venue availability.

3. The user can interrupt the system while it is scheduling.

Scheduling software must be 'interactive' in at least one of the senses above, since no software package can schedule a sporting season that will perfectly suit all the conflicting aims and restrictions imposed by the competing teams, venue managers, sponsors, supporters and television networks. In this thesis a combination of the second and third usage is used: the software should be responsive, so that it can provide a number of different schedules in a small amount of time. Further, the user should be able to use an existing fixture as a starting point for further (automatic) refinement. Finally, the user should have
the ability to pause the algorithm at any point, analyze the best schedule generated so far, and then add or remove constraints as they see fit. This approach allows sports league officials to 'tweak' input parameters to the scheduler to achieve preferable outcomes, or to make minor changes to the fixture by hand.

Finally, the reader should note that this thesis does not describe the development of a commercial product, and so details of, for example, a user interface, are absent.
Chapter 2: Literature Survey

2.1 Introduction
This literature survey examines:

a) Genetic algorithms, particularly with respect to recent advances, and
b) Techniques that have developed, both GA and non-GA based, to solve
   scheduling and timetabling type problems.
c) Combinatorial issues with respect to scheduling round robin tournaments.
d) Available software packages for scheduling sporting competitions.

2.2 Genetic Algorithms
This section discusses (briefly) the genetic algorithm approach, including some
of the more advanced techniques that have been developed in recent years.
This discussion is intended to provide a background to the reader; more detailed
introductory information can be found in [Goldberg].

GAs are often used for several reasons. Amongst these are:

1. They are efficient. Theory suggests [Goldberg, p41] that the ‘building
   blocks’ (more formerly: ‘schema’) of a good solution are sampled in an
   inherently parallel fashion; indeed, a population of size N will sample \(O(N^3)\)
   schema per generation.
2. GA’s are domain independent (they are a ‘weak’ method), and so can be
   applied quite flexibly.
3. The search space is widely sampled by the population of (usually) randomly
   generated individuals. This helps prevent the search being trapped in local
   minima.
4. Competing criteria can be combined in an objective function, with
   weightings applied accordingly. Further, by maintaining a population of
   solutions, ‘genetic algorithms can search for many non-inferior solutions in
   parallel’ [Fonseca & Fleming, p1].
2.2.1 Simple Genetic Algorithms (SGA)

Genetic algorithms attempt to mimic the success of natural evolution by loosely emulating its mechanisms. Essentially, a population of potential solutions is maintained, and the fittest individuals in the population survive and reproduce, thereby creating the next generation of (hopefully) fitter individuals. The algorithm itself is domain independent.

The simple GA (sometimes called the 'canonical' GA) uses three primary genetic operators: Reproduction, Crossover & Mutation [Goldberg, p10]. The reproduction operator selects a subset of individuals from the population, such that individuals with a higher fitness have a greater probability of being chosen. Fitness is determined by an objective function, which must be written by a domain expert.

Individuals who have been selected for reproduction create two offspring (per couple) by exchanging genetic material. The domain expert selects the form of the genome containing this genetic material, which is chosen to minimize individuals that are 'lethal' [Goldberg, p78] i.e. violate hard constraints. The SGA limits the form of the genome for simplicity - attributes such as diploidy (i.e. pairs of chromosomes) are not usually employed. The exchange of genetic information is via single point crossover. This means that a point is chosen, at random, along the genome, and the first child receives the genetic information from the first parent before the crossover point, and receives the remainder from the second parent. The second child is created from the opposite information. Crossover attempts to propagate high quality 'building blocks' of genetic information to successive generations, thereby eventually creating individuals with many good qualities.

Finally, mutation is randomly applied to the children to prevent premature convergence of the population. Mutation is an attempt to balance exploration of the unknown areas of the search space and exploitation of the areas that are known to be better than the average of the current generation.

---

1 Avoiding premature convergence is important, since otherwise the risk of finding a locally (not global) optimal solution is increased.
Selection is often implemented with a 'roulette wheel' technique, where the slots in the roulette wheel are in proportion to the fitness of the individual [Goldberg, p11]. Common improvements to the basic algorithm include elitism, which ensures that the best individual always survives to successive generations, and either two point or uniform crossover. Uniform crossover operates by flipping a coin for each gene in the chromosome to determine which parent the gene will come from. While uniform crossover would seem to be highly disruptive to genetic building blocks, several researchers have shown empirical benefits; [Whitley 1994] presents several reasons why this may be so, including the disruption preventing premature convergence of the population.

2.2.2 Maintaining Diversity

The problem of premature convergence of the genetic information within a population has long been recognized, and numerous methods have been put forward [Goldberg]. The most obvious method is mentioned in the previous section - the mutation operator. Other methods include 'crowding', whereby new individuals replace those most similar to themselves in the population. Note that crowding should not be confused with 'speciation', which is an attempt to force the GA to find multiple optima [Goldberg, pp. 185-197].

Crossover may also be used to maintain diversity [Jones]. In this paper the "mechanics" and "theory" of crossover are compared. Jones describes the theory of crossover by stating that fit individuals (relative to the population average) have good genetic building blocks, which, when combined with good quality blocks possessed by other individuals, will probabilistically produce even fitter offspring. Jones investigates the mechanics of crossover by mating individuals with randomly generated genomes, thus preventing the removing of the 'good building block' hypothesis from the equation. Interestingly, he found

---

2 Another form of this method is when children to replace their parents in the population [Cavicchio].

3 Speciation is often implemented by lowering the fitness ratings of individuals who have similar genetic material.
little difference in performance when using randomly generated partners or partners selected using a roulette wheel method, using a variety of test functions. This lead Jones to conclude that the disruptive nature of crossover is more important than its ability to combine building blocks.

The CHC algorithm [Whitley, p30] uses 'cataclysmic mutation' when the population converges, which means that every individual (except the best) suffers dramatic mutation.

'Incest prevention' is another method used by CHC to encourage diversity; here individuals are not allowed to mate with each other if they are too similar.

**2.2.3 Stopping a Genetic Algorithm**

In many problem domains (including ours), we do not know a priori what the optimal solution is. The GA research community often simply restricts the number of generations, but may also use a measure of the convergence of the population. There is more than one method used to check convergence, and two that are often employed [Wall] [Schraudolph & Grefenstette] are to check the score of the best individual over a number of generations, or the percentage of individuals in the population that are identical (or similar) to each other.

There does not seem to be any real consensus in the literature as to the best approach, and investigation seems to be of low priority (there are few papers on this issue). This is perhaps because the stochastic nature of genetic algorithms invites the researcher to run a given algorithm a number of times, and so we are given a 'feel' for the best achievable solutions.

Nevertheless, the GA community has given some thought to the topic. It has been noted [Whitley, Beveridge, Graves & Mathias] that although the issue has been largely ignored by the literature, a GA can typically waste 50% of processing time driving the algorithm to convergence after the best solution has been found. They suggest using domain knowledge to formulate 'Intelligent stopping criteria'. Unfortunately, they do not demonstrate using such criteria.
2.2.4 Replacement Methodologies

Steady-state algorithms generate only a small number of offspring in each generation. This leads to the question of which current member(s) of the population should be replaced. The GENITOR algorithm [Whitley, p29] replaces the worst member of a population, thereby allowing the GA to become more efficient. Poor individuals are removed from the population faster and the average fitness of the population within the GA can never decrease from one generation to the next (i.e. the algorithm is monotonically improving). The CHC algorithm [Whitley, p30] uses a variant of worst member replacement, by assessing the quality of individuals in both the old and new generation, after removing duplicates.
2.2.5 Advanced Selection Methodologies

2.2.5.1 Rank Based Selection
Rank based selection [Baker] orders all individuals according to the ‘raw’ value returned by the objective function, rather than their fitness value (i.e. no scaling is used). The best individual is given the largest proportion of mating opportunities (the exact proportion is chosen somewhat arbitrarily), while the worst individual is given the least number of mating opportunities.

2.2.5.2 Tournament Selection
Tournament selection [Goldberg, pp. 121] randomly selects (with replacement) two individuals from the population. These individuals are participants in a “tournament”, where the individual with the highest fitness wins. The prize for the winner is entry to the reproduction pool. The process is repeated until enough individuals have been chosen to reproduce the next generation. Tournament selection has a ‘bias’ of two [Whitley, Beveridge, Graves, Mathias] since the expected number of contests per individual is two, and the fittest individual will win both contests.
Benefits of tournament selection include easy implementation on parallel computing architecture, and that a sorted population is not required.

2.2.6 Crossover and Mutation
Using genetic algorithms for timetabling is similar to the TSP or JSSP (see section 2.3.3), as it often involves the use of order or position dependent genomes, since the optimum or best sequence of activities is sought. Hence, an illegal solution may have the same value multiple times in the genome (“superposition”) and be missing other values. Techniques that prevent creation of these 'lethal' individuals are important for the efficient execution of a GA, and are presented in the subsections that follow.
2.2.6.1 Permutation Based Crossover Techniques

Several permutation based crossover techniques have been used for ordered genome encodings, the most popular being partial match (PMX), order (OX), cycle (CX), maximal preservative crossover (MPX) and edge crossover. The first three are described in [Goldberg p.171-175] and [Whitley & Yoo], while MPX & edge crossover [Mathias & Whitley] are operators especially designed for the TSP.

Order crossover is described in detail here because it is selected for use later in this thesis.

1. Call the two parents A & B.
2. Two points \((p_1, p_2)\) are selected at random, with \(0 \leq p_1 < p_2 \leq l\), where \(l\) is the length of the chromosome.
3. The points \(p_1\) & \(p_2\) are used to define a contiguous section \(s\) in A.
4. The first child, \(A'\), inherits all gene values from B in the same position as they appeared in B - other than those genes with identical values to those in \(s\). This results in \((p_2-p_1)\) gene positions without values in \(A'\).
5. Beginning at \(p_2\) in \(A'\), genes values are shifted left to fill gaps; end values are filled from the start of \(A'\). At this point a contiguous section \(s'\) exists in \(A'\) where \(\text{length}(s) = \text{length}(s')\) commencing at \(p_1\) within which no gene has a value.
6. The section \(s\) is copied from \(A\) to \(A'\).
7. The other offspring is generated in a complementary manner.

2.2.6.2 Genetic Repair Operators

'Normal' (i.e. single or multiple point, or uniform crossover) can still be used with genetic encodings without flooding the population with 'lethal' individuals if genetic repair operators are used. These operators attempt to correct the genes to make sure there are no occurrences of the same value twice in the string. However, the repair operation has no recognition of the building blocks within the string, and is therefore unlikely to preserve them [Norenkov & Goodman].
For this reason, most researchers prefer avoidance methods [Bruns], [Syswerda & Palmucci].

2.2.6.3 Permutation Based Mutation
Swap mutation is a unary operator; one individual is selected at random, and then two positions within the genome of the individual are chosen at random and their values switched. Note that no new illegal values can be introduced, and no values can be eliminated or duplicated.

2.2.7 Hybrid Genetic Algorithms
A hybrid GA is one that employs a GA in conjunction with a non-GA technique, and such techniques frequently outperform a pure GA. The non-GA technique is often domain specific, and attempts to maximize the exploitation of good individuals, while the GA is used to efficiently explore the search space. This technique may be a hill climbing or gradient ascent method, and is activated as each new individual is created. However, 'smart' crossover techniques may also fall into this category if the crossover method is specific to a given problem domain.

When considering results from a hybrid GA it is important to note that the execution time for each generation may be significantly longer, and so comparisons should not be made on the number of generations alone.

2.2.8 Deterministic Genetic Algorithms
One drawback of GAs that seems to be rarely considered is that the same individual (or different individuals with the same genetic material) may be assessed by the objective function many times because no records are maintained of the individuals assessed. [Salomon] suggests that removing this inefficiency can optimize the performance of a genetic algorithm, which he submits is due to the random nature of GA operators. Instead, his methodology

---

4 Such techniques are often said to employ 'Lamarckian' evolution.
5 No records are kept because of the potentially enormous quantity of memory required.
does not use a crossover operator, and always applies mutation, in the process removing resampling\(^6\). Salomon criticizes test functions used in GA literature as having little interaction between variables, and refers to theory that suggests that in the presence of epistasis\(^6\) GA performance is actually worse than random search. He proposes his deterministic GA as an improvement on current GA research.

\(^6\) See glossary.
2.3 Scheduling Algorithms

This section explains some of the approaches used to automate scheduling problems, but concentrates on those that employ a GA.

Scheduling problems are typically NP-Complete [Garey & Johnson] and so there are no known algorithms that can find an optimal solution in polynomial time. Therefore, for sizeable scheduling problems finding the optimal solution may not be practical, and we resort to techniques that find reasonable solutions in an acceptable timeframe.

2.3.1 Round Robin Sports Scheduling

The literature specifically related to round robin sporting competitions seems relatively sparse, particularly when one considers the very large variety of constraints that may be imposed by different problems in this field. In particular, we did not find any attempts to apply GAs to sports scheduling.

Henz considered the problem of scheduling the Atlantic Coast Conference basketball competition. The constraints used in this problem included:

- Return match separation: there is a minimum number of rounds allowed between the first and second time two teams meet.
- Final Aways: teams cannot play away on both the last two rounds.
- Rivals: Every team (except ‘FSU’) must play against its traditional rival in the last round.
- Opponent ordering: No team can play in two consecutive rounds against ‘UNC’ and ‘Wake’.

The optimization criteria (which we refer to as ‘soft’ constraints) included:

- Avoid two opening away matches.
- Reduce number of consecutive home matches (including byes as home matches).
- Dates are pre-ranked by venue.

It is stated that the optimization criteria are conflicting, and therefore require a pareto-optimal [Fonseca & Fleming, p1] solution. No provision is made to rank
these conflicting criteria, and so each timetable produced must be manually assessed.

The problem is to generate a double round robin. ‘Mirroring’ is defined to mean that games in the first round robin are repeated in the same order in the second half of the season, except that home and away teams are switched in each game. Perfect mirroring was not possible due to constraints, and so some rounds were manually switched.

Teams are assigned to patterns using the constraint programming language, Oz. A ‘pattern’, as used by Henz, is a list of H,B, and A characters denoting Home, Bye & Away games, respectively. It is shown that tighter constraints reduce the number of possible tournaments (‘pattern sets’), and so speed the execution of the system. Patterns are first assigned to teams so that constraints related to each team can reduce the possible number of pattern sets. Secondly, opponent teams need to be assigned to matches, and constraints are employed to prune the search space. Henz reports very fast run times compared to an earlier paper using similar techniques: 117 seconds compared to 24 hours, with 125 solutions of higher quality.

Schaerf [Schaerf 1996b] uses constraints from the European soccer leagues, such as the Italian “Serie A” national competition. Some of the more interesting constraints from this paper are shown below:

- Mating: in a given round a pair of teams may be barred from playing each other. One reason why this is used is to prevent top teams playing each other before international matches.
- Triples: prevents three selected teams from playing simultaneously in a geographically nearby area, so that trains, highways etc are not overcrowded.
- Grading: a small number of teams shall be specified as the best in the competition. The schedule will be optimized so that the time between playing two of these top teams is greater than a certain number of days.
- Availability: some venues may be unavailable because of other bookings.
Scheduling is performed in two steps. Firstly, Schaerf cites [de Werra], who first proved that there exists a minimum number of breaks [see glossary] for a round robin tournament (2n-2 breaks for a single round robin tournament, where there are 2n teams), and then gave a tournament pattern for this minimum. A pattern specifies which games are played in each round, including which team plays at home. The paper uses a modified version of this pattern for double round robin tournaments that minimizes breaks at the middle of the season. In a second step, teams are assigned to the pattern: a bipartite graph matching problem, made complex by the need to satisfy the various constraints. Using the mating constraint, Schaerf shows that the matching of teams to the pattern is NP-complete. Further, the decision problem “Does a tournament satisfying all hard constraints exist” is also shown to be NP-complete [Schaerf 1996b, page 8]. This was obtained by reducing the known NP-complete decision problem “Is a partial tournament not premature?” to tournament scheduling with mating constraints. The ECLiPse finite domain library is then used to solve the matching problem.

The results obtained indicate a wide range of solution times, depending on the number of teams and the types of constraints imposed. For a twelve team competition, the optimal schedule was found in seconds. For eighteen and twenty team competitions, solutions were found between five (tightly constrained) and sixty minutes (loosely constrained).

Schaerf makes an important note: “it is not known how to enumerate all possible patterns in a computationally tractable way” [Schaerf 1996b, page 8].

2.3.2 The School Timetabling Problem

In many respects, scheduling teacher, class and room timetables for a high school or a university faculty is very similar to the sports scheduling problem, and there is a significant body of research in this area.

---

7 A premature tournament is a partially scheduled tournament that cannot be completed without violating hard constraints.
A comprehensive survey of this field can be found in [Schaerf 1995]. Schaerf defines timetabling as "consisting of fixing a sequence of meetings between teachers and students in a prefixed period of time... [while] satisfying a set of constraints". He then states that nearly all variants of the timetabling problem are NP-complete, and concludes that only heuristic methods, which do not guarantee an optimal solution, are practical. The heuristic methods discussed included Genetic Algorithms, Tabu Search, Simulated Annealing, Direct Heuristic Methods, Network Flow Techniques and Graph Colouring. Integer linear programming, constraint logic programming (CLP) (see also [Henz & Wurtz] (the Oz language)) and constraint relaxation methods (also see [Boizumault et al]) are also mentioned. While the author did not conclude in favour of any one method (because of the varied testing methods), it is stated that:

- The results are nearly always superior to those generated by hand
- The results were usually produced in the order of a few seconds, on hardware ranging from PC to mainframe.
- It is very difficult to write software that can be used by different institutions.

In a separate paper [Schaerf 1996b], Schaerf discusses the application of the Tabu Search technique to high school timetabling problems in detail. Tabu search is similar to hill climbing techniques, as it uses the concept of a 'neighbour' in the search space. The neighbour with the best evaluation is the new solution, while maintaining a fixed length list of 'tabu' locations prevents cycling. The effectiveness of any algorithm that uses the notion of neighbours depends the encoding of the search space. Schaerf used a representation previously used for genetic algorithms [Coloni 1992a], which used a table, each row representing the timetable of a teacher. Schaerf's results state that, over a number of different schools, the tabu search method improved on results obtained by hand, and a feasible solution was always found within minutes on an SGI-Indy workstation.
School timetabling using a parallel genetic algorithm is discussed in [Abramson & Abela], where an objective function is used that simply finds where any clashes occur - i.e. the authors are trying to find a feasible (not necessarily optimal) solution. A multiple chromosome encoding is used, where each period is a chromosome containing a number of \{class, teacher, room\} tuple identifiers, and different chromosomes may have a different number of tuple identifiers (clearly, there are a fixed number of possible tuples). The crossover operator was implemented as a fairly standard single point crossover (with repair), while mutation took advantage of the multiple chromosome encoding by switching a randomly chosen tuple identifier from a randomly chosen period to another randomly chosen period. The parallel nature of genetic algorithms is exploited by using several concurrent processes to operate on a fixed percentage of the population. Feasible solutions were found for all data sets presented and an almost linear reduction in execution time (with respect to the number of processors used) found.

In [Colorni 1992b], multiple chromosomes are again used, but each chromosome is the timetable for a teacher, and the chromosomes are fixed length, with each position in a chromosome corresponding to a teaching period. Simple single point crossover is again used, and again genetic repair is required to remove the infeasible solutions created. Two mutation operators are used; the first shifts several consecutive hours (chosen from a random location) for a randomly chosen teacher to a randomly chosen destination (in the same teacher's timetable). The second operator is a special case of the first: whole days are switched. Unfortunately, few results were presented in this paper.

[Weare et al] present a hybrid genetic algorithm for timetabling exams at universities, using heuristic crossover operators and seeding of the population. The authors state that the problem can and is solved, but only with 'huge amount of administrative effort'. Their approach is to ensure that no infeasible
solutions (i.e. too many exams for the number of rooms, or students with exam clashes) can be generated by encoding the problem carefully, and by using special purpose crossover and mutation operators. In this case, the crossover operator takes exams for a given time from each parent, and then adds exams to that time (period) according to a heuristic, making sure that no new exam conflicts with those already present. The heuristics tested included random selection, graph colouring heuristics, and even smarter rules that, for example, checked whether there is a conflicting exam in the previous period (this endeavored to spread students exam load across the exam period). The number of periods was not fixed, and so the objective function was used to reward shorter examination timeframes.

Although the authors results show excellent performance, such a method is unlikely to be of assistance in the sports scheduling domain, because it is not as tightly constrained: in a common twelve team competition, there are only six games per round, and the length of the season is fixed.

[Salwach] investigates the school timetabling problem, specifically the problem of scheduling n classes over 35 hours (7 hours per day), with a predefined list of subjects that every class must take. The problem of assigning these classes to rooms was not considered. The encoding used was as follows:

- Each class (i.e. group of students) has an array, of length 35.
- Each position of each element in the array is important: it is the hour number. For example, position 1 is the first hour on Monday, and position 8 is the first hour on Tuesday.
- The value of each element represents a teacher/subject pairing.

Salwach uses a simple GA with a permutation based crossover (PMX, OX & CX) and swap mutation. Along with some problem specific soft constraints, the obvious hard constraints are present: a teacher can only be with one class at a time, and a class cannot have more than one teacher at a time. Note that the encoding satisfies the
second constraint, but the first constraint must be satisfied explicitly by the objective function. Since the search space is quite large \((35!)^n\), we might expect the 'lethal' individuals (caused by too many teachers in a class at one time) to prevent efficient execution. Indeed, Salwach finds that just four classes\(^8\) \((n=4)\) prevents the GA from converging.

Salwach proposes a merge of classical and GA search techniques: reduce the search space, and when successful, iteratively expand the problem to its full size. If the larger problem again cannot be solved, backtrack to the smaller problem and try again. The problem was reduced by shortening the number of hours to in the week to seven, and then expanding to 14, 21, 28, and finally, 35 hours. The objective function is also made more sophisticated: if a trial solution of the sub-problem indicates that a corresponding solution to the full problem is likely to violate constraints, then a penalty is applied. The value of the penalty depends on how likely the violation is to occur. Note that the objective function now incorporates a heuristic.

The changes produced far better results: the 4 class problem was solved in only 30 generations, while the 8 class problem was solved in 500 generations. However, the 12 class problem was still unsolvable.

### 2.3.3 Job Shop Scheduling

Scheduling using Genetic Algorithms is discussed in more detail in [Husbands], with particular reference to the Job Shop Scheduling Problem (JSSP).

The JSSP problem is summarised as follows: a number of 'jobs' need to be done on a certain number of machines; the time needed to complete the jobs is to be minimised, subject to certain constraints. The time required is known as a makespan. Husbands states that most scheduling tasks are quite similar in format to the JSSP, and then outlines instances of GA research into such related problems, including bus driver scheduling, packing problems (e.g. pallet loading) and layout problems (e.g. VLSI layout). The author concludes by stating that if

\(^8\) Salwach states that eighteen classes is a 'real world' figure.
deterministic methods are appropriate they should be used; GAs, however, are useful where the problem is difficult to formalize.

In [Norenkov & Goodman] the NP-Complete nature of scheduling problems is recognized, and the Heuristics Combination Method (HCM) is presented as a possible solution and applied to a job shop scheduling problem (JSSP). HCM is a philosophical shift from more traditional genetic encoding approaches, in that gene values represent heuristics (e.g. choose the fastest machine), rather than something related directly to the problem (e.g. in the JSSP, a machine). The objective is to find the optimal sequence of heuristic rules. Since we may wish to apply these rules many times in a sequence, permutation crossover techniques and genetic repair are not required - n point crossover can be used efficiently.

After investigating several JSSP type problems, the authors conclude that complex scheduling problems are amenable to solution with HCM if the subset of heuristics provided to the algorithm is rich enough.

2.3.4 Airline Crew Scheduling

[Levine] studies the Set Partitioning Problem (SPP), and applies it to the problem of assigning airline crews (pilots and attendants) to 'legs' (a takeoff and landing) in the schedule of an airline. The SPP is visualized as a table, where each element may take on a binary value. In this case, each row of the table is a leg, and each column is a crew; an element is true (binary 1) if the crew represented by the column will travel on the leg represented by the row. This is a particularly large problem: the author states 'problems with hundreds of thousands of columns are considered very large'. Levine investigates sequential, parallel and hybrid GAs, compares the simple GA with a steady state GA, and also experiments with GA parameters, such as the selection algorithm.

2.3.5 Scheduling with Graphs

Perhaps the first attempt at solving the scheduling problem over a range of problem domains was with the use of mathematical graphs, and this technique
is still widely studied. An example of this is in [Kiselyov], where a chess tournament was scheduled by considering each vertex in the graph as a game between two players; thus there are \(n(n-1)/2\) vertices in a graph, where \(n\) is the number of players in the tournament. The vertices are connected to other vertices if the games represented by the vertices involved could take place in the same round of the tournament. Simple constraints (e.g. A player can only play one game per round) are satisfied implicitly by this representation, whereas higher level (and less critical) constraints (e.g. Players should play opponents of roughly equal ability) is satisfied by associating a cost function with each vertex.

Finding the best schedule involves finding a set of vertices that are connected to each other, with the lowest cost. Unfortunately, this itself is a NP-complete problem, and so Kiselyov resorted to the less complex (and therefore faster) task of finding partial schedules. This was rejected because of the amount of memory consumed by the algorithm. Kiselyov then used a domain specific ‘greedy’ strategy, which was not guaranteed to find the optimal solution, and did not utilize the graph method.

2.3.6 Interactive Scheduling

Nearly all work in the automated scheduling field is ‘batch’ style (see Glossary); constraints are specified and then classified into soft and hard constraints, which are often hard coded into the software.

[Poeck and Hestermann, p1] suggests extremes for interactive scheduling:

"In one extreme the human planner may completely construct the schedule in an interactive graphical environment whereas the system only checks the production requirements... to totally automatic plan generation. In addition... The person planning interactively is supported by the system which generates suggestions like e.g. a suitable resource and time-slot for a chosen activity. The automatic planning is capable of
The authors suggest a four step process to scheduling (select activity, select resources, test constraints, attempt heuristic repair), in which each step can be performed by the user or the software. Unfortunately, this approach is not well suited to a GA based system, since it (the four step process) is performed on a single individual.

Interactive scheduling relating to round robin sporting tournaments is discussed in [Schaerf 1996b, page 13], where a pseudo-code algorithm is given for such tasks. Briefly, an approximate solution is found, and then the user is allowed to adjust the problem specification. Subsequently, a local search algorithm is used to adjust the solution.

In [Boddy et al] work is reported on scheduling problems at NASA. This paper states that the large number of constraints make the use of OR type algorithms (e.g. linear programming) impractical. Further, a process of 'iterative refinement' is used to support the process by which scheduling is undertaken. In this process constraints are gradually added (or removed) during scheduling, thereby indirectly fixing activities in the schedule. Their prototype system is an extension of an existing system, known as the Time Map Manager (TMM). The TMM allows users to "assert constraints between temporal points" and performs causal reasoning based on these constraints. As this system is quite complex, and the approach used is from a completely different viewpoint to that suggested in this thesis, it will not be considered further.

2.4 Combinatorics of Round Robin Tournaments

2.4.1 NP-Completeness

Problems in ‘P’ are defined as those solvable in polynomial time. Problems in ‘NP’ are also solvable in polynomial time, but this is only the case [at the time of writing] when a non-deterministic algorithm is used. A non-deterministic algorithm is one where an ‘Oracle’ makes the decisions on which path in the
search space to follow. An Oracle always knows the best path to follow due to some magical prescient power.

Polynomial algorithms are those that solve the problem in \( O(N^K) \) time or faster, where \( N \) is the size of the problem. Algorithms worse than this (including those solved in \( N! \) time) are called ‘super-polynomial’ or exponential\(^9\) [Harel, p. 157].

Problems can then be divided into ‘tractable’ & ‘intractable’, depending on whether they admit deterministic polynomial time algorithms. No NP-complete problem has been proven to require super-polynomial time (in the worst case). Furthermore, every NP-complete problem is reducible in polynomial time to the other NP-complete problems, so that if any one problem were shown to be solvable in polynomial time, all those problems would also be ‘in \( P \)’. This grouping of problems gives the term ‘complete’. Not all problems in NP have also been shown to be NP-complete.

To prove an algorithm is NP-complete, there are two steps:
1. Show the problem is in NP.
2. Show a known NP-complete problem can be reduced to this problem in polynomial time (i.e. it is NP-Hard).

### 2.4.2 Genetic Algorithms & NP-Complete Problems

While NP-Complete problems can only be guaranteed to be solved correctly in super-polynomial time [again, according to current theory], there are a class of algorithms known as ‘weak’. Such algorithms may occasionally find an incorrect (or in our case, sub-optimal) solution, but good algorithms in this class will usually find the correct solution in a much faster time period. These algorithms make use of probabilistic methods (Harel, p. 308).

Genetic Algorithms use probabilistic methods (e.g. initially random population, probabilistic selection routines, etc.) in an attempt to solve otherwise intractable problems. Since GAs use a population of solutions, it is also reasonably safe to assume that while the very best solution may not be found, a

\(^9\) Note that the so-called exponential algorithms also include \( O(N!) \) algorithms and others.
close to optimal conclusion is highly likely\textsuperscript{10} in much better than exponential time. Further, the chance of finding a poor solution can be reduced if a number of simple techniques are used, including running the algorithm several times, increasing the population size and so on.

\textbf{2.4.3 Latin Squares}

A Latin Square is an $N \times N$ matrix of integers, where each integer is in the range $[1,N]$, and no integer appears twice in the same row or column. These squares can be used to construct round robin tournaments [Bogomolny] as follows.
1. Let $N = 2n - 1$, where $2n$ is the number of players\textsuperscript{11}.
2. Construct a $N \times N$ magic square, ensuring that no number in the left top to bottom right diagonal is repeated.
3. Construct a new column (column $N+1$) and a new row (row $N+1$) by copying the values at the diagonal to the new column and row.
4. Remove the diagonal values.

The result is a grid of numbers representing the rounds in which players play each other. For example, if the value at position $\{1,3\}$ is 3, then round 3 contains a game between players 1 & 3. An example from [Bogomolny] is repeated below for $2n = 6$ players:

\begin{table}[h]
\begin{center}
\begin{tabular}{cccccc}
1 & 2 & 3 & 4 & 5 \\
2 & 3 & 4 & 5 & 1 \\
3 & 4 & 5 & 1 & 2 \\
4 & 5 & 1 & 2 & 3 \\
5 & 1 & 2 & 3 & 4 \\
\end{tabular}
\caption{Latin Square with Unique Diagonal}
\end{center}
\end{table}

\textsuperscript{10} This depends, of course, on the size of the population relative to the size of the search space, the GA’s termination condition, and perhaps on how misleading the search space is.
2n ensures an even number of players. An odd number of participants simply requires a ‘phantom’ player to represent a draw.
If a round robin tournament with 2n players is arranged into (2n-1) rounds of matches (i.e. each player plays every other player exactly once), then we can construct a tournament using latin squares as shown in the previous section. The number of different latin squares of size m x m (m=2n-1) is given by [Bogomolny]:

\[ N(m) = m!(m-1)L(m) \]

Where \( L(m) \) is the number of 'normalised' latin squares (the first row is in ascending order). \( L(m) \) is calculated numerically; for \( m=12 \), \( L(m) = 1.62 \times 10^{44} \). These calculations suggest that it is very difficult indeed to find the optimal solution to a scheduling problem, since the search space increases exponentially with the number of teams or players: the problem appears intractable. However, the formula misses one important point: the number of Latin squares is reduced significantly by the constraint that the diagonal must not contain duplicates.

To our knowledge, the number of latin squares with unique values in the diagonal is not known. We used a backtrack algorithm to try to find this computationally, but (not unexpectedly) for even an eight team competition this proved an intractable task for PC hardware. [McWorter] provides further information on this subject:
“I believe nothing is known about the number of round robin tournaments. Certainly a given tournament can have a number of permutations, but if (a) the order of games in a round is not important (b) the number of teams is n then the number of pre-schedules is \((n-1)!f(n)\), where \(n\) is the number of teams, \((n-1)!\) is the number of permutations of the rounds, and \(f(n)\) is the tricky, and unknown, number of inequivalent tournaments.”

12 McWorter uses the term ‘pre-schedule’ to indicate a list of games that have no dates or venues assigned.
2.5 Software Packages for Sports Scheduling

For a complete review of scheduling algorithms, we shall examine commercially available systems.

The author reviewed two commercially available software packages: Schedule Wizard for Windows\textsuperscript{13}, and the Team Sports Scheduling System\textsuperscript{14} (TSSS).

Both offered the following features:
1. Automatic generation of fixtures.
2. Venues can be unavailable on given dates.
3. Teams can be unavailable on given dates.
4. Teams can have home venues, or all teams can play at a venue.
5. Teams can be split into divisions. For example, a league may have 'North' and 'South' divisions; all teams within a division teams play each other twice, and play teams outside the division once.
6. Both packages have a simple Microsoft Windows based user interface.

In addition, the TSSS product allowed fixed date games to take place, by allowing the user to reschedule specific games after the automated scheduling finishes.

However, both packages had the following limitations, making them unsuitable for use for more difficult problems (including the NBL):
1. Teams cannot have games on preferred weekdays, unless all games are on that weekday.
2. Road trip pairing of teams cannot be specified.
3. There is no explicit specification of the number of allowed consecutive home (or away) games.

The companies responsible for these products would not release details of their scheduling algorithms.

\textsuperscript{13} Timeless Technologies Inc.
\textsuperscript{14} T & C Software Inc, http://www.oz.net/~tcs
Chapter 3: The Domain

3.1 Introduction
When encoding a problem for genetic based search, we want to minimise the number of infeasible individuals that can be represented. This indicates that we should separate the problem constraints into two distinct categories: ‘hard’ constraints, which if breached allow very poor (infeasible) solutions, and ‘soft’ constraints, which allow grading of feasible solutions.

This chapter lists the constraints in the problem domain, separating them into the said categories, and then examines encodings that will assist or prevent the genetic algorithm from generating infeasible solutions.

3.2 Problem Specification
Since there is a wide variety of different types of sporting organisations, we must categorise the problem domain. In essence, the domain has the following features:

- Venues are not automatically available – bookings are required.
- Team travel costs are important – the teams may be dispersed over a wide area.
- Revenue must be maximised – gate takings and television coverage are important.
- The fixture must be fair – no team can be disadvantaged.

3.3 Constraints Specified by the NBL
The National Basketball League specified the constraints listed in this section. They are used in this thesis to ensure the research has a solid grounding in reality.

The NBL currently consists of 11 teams, with two more expected to join in season 1998. After meeting with the NBL, the following constraints were identified, with hard constraints in bold print:
1. Each team must play every other team exactly $n$ times during the season, where $n \geq 1$. Following from this, each team must play the same number of games over the duration of the season.

2. A venue can have one game per day.

3. Games in successive rounds may not be closer than 24 hours apart (to give players time to rest).

4. Some date/venue combinations are not allowed, due to bookings made by other events (e.g. rock concerts, tennis tournaments).

5. Some games have fixed dates and venues. These games are specified in advance by the NBL to maximize revenue from the games.

6. Every team must have the same number of home games.

7. No team can play another twice at home (or twice away) in a season where each team plays every other twice.

8. Each team must play exactly once per round.

9. Friday & Saturday night games should be maximized (larger attendances on these nights).

10. Teams should be paired to minimize travel. E.g. Townsville/ Brisbane; Adelaide/ Perth. That is, when a team plays Brisbane at Brisbane's home venue, they should play their next game against Townsville at Townsville's home venue. This is known as a 'Road Trip'.

11. Games played during a road trip should not be too many days apart - teams do not wish to stay for longer than necessary in hotel rooms.

12. No team should have more than three successive home games (to maximize attendances).

13. Some date/venue combinations are not preferred, due to clash with another major event (e.g. AFL football)

Additionally, the solution must be reached in 'reasonable time' on readily available computer hardware (e.g. Pentium based PC). 'Reasonable time' is taken to mean minutes, rather than hours. This is to allow generation of a
number of alternative fixtures for review by a sporting administrator, who typically does not have access to high end computational power.

### 3.4 Further Assumptions and Constraints

The assumptions and constraints listed below have been chosen based on the authors' experience of sporting competitions - they were not explicitly mentioned by the NBL.

- Each team has a home venue; teams can share venues.
- Each team must play the same number of home games.
- Rounds are played on a weekly basis.
- In the event that the preferred game day is not available, weekend games (Fri/ Sat/ Sun) are preferable.

### 3.5 Discussion of Constraints

#### 3.5.1 Other Basketball Competitions

To ensure the resulting system is, at the very least, general enough to work with other large scale basketball competitions, we shall look at the Atlantic Conference U.S.A Collegiate basketball's competition, as reported by [Amzi].

1. Each team must play each other team twice, once home and once away.
2. Some teams must play on certain [week] days.
3. Teams shall have equal home and away games during each half season.
4. Teams shall not have more than two consecutive home or away games.
5. Additional constraints based on TV coverage. [This is interpreted as meaning some games must have fixed dates].

The third constraint is the only one not directly specified by the NBL. Also see section 2.3.1, page 13 for a brief discussion of additional constraints related to European soccer competitions.

#### 3.5.2 Appropriateness

The encoding will not try to be ‘all things to all people’. In particular, the encoding is not designed for local sporting associations, and so constraints
related to these types of competitions are ignored. This decision was made as there is good support offered by commercial software packages [see section 2.5, p. 28], and no obvious need for research.

3.6 Problem Analysis

The mathematical field of combinatorics provides insight into the computational complexity of the problem domain [see section 2.4, p. 22]. Here we show the NP-completeness of the problem: “Can a tournament schedule be created which does not violate any of the hard constraints?”.

Firstly, the problem is in NP, because it is clear that the constraints for a guessed tournament timetable can be verified within polynomial time.

Secondly, we have to show the NP-Hardness of the problem. In [Schaerf 1996a, page 8], it is proved that scheduling a round-robin tournament with mating constraints is NP-Complete. It is clear that mating constraints can be reduced to constraint 4 (section 3.3, page 29) in polynomial time, and so we can conclude that the problem is indeed NP-Hard and also NP-Complete.
Chapter 4: The Algorithm

4.1 Introduction
In this section we discuss two possible genetic encodings, and show how tournament schedules can be created and evaluated using them. The first encoding presented is shown to illustrate why a more complex approach is required.

4.2 The Encoding

4.2.1 A Naïve Encoding
The first encoding considered (but subsequently found lacking during experimentation) was to enumerate all the games to be played, and then assign a position to each of these games in the genome. The value at each position is a date identifier; the identifier can then be used to find the actual (calendar) date from a list maintained for each venue. For example, if position $n$ in the genome represents a game between teams X & Y, and the value at $n$ is $D(n)$, then the game date is found by looking up the list of available dates at team X’s home venue\(^{15}\) using $D(n)$. The encoding is represented pictorially below:

\(^{15}\) Team X is at home because of convention: the first team named is the home team.
This encoding seems attractive since it allows 2 point or uniform crossover, both of which are much less computationally expensive when compared with permutation type crossover operators. Furthermore, it is clear that the encoding allows for representation of every possible tournament.

A good encoding will minimize or eliminate the generation of individuals that are impossible to use in reality (i.e. violate hard constraints). This encoding deals with the hard constraints (section 3.3, page 29) as follows:

- Constraint 1: Eliminated, since all games are enumerated in the encoding.
- Constraint 2: Not eliminated: since alleles represent indices into an array of available dates, and two alleles may have the same value, a venue may be double booked. The alternatives, such as removing dates from the list of those available for each venue as they are booked (effectively giving priority to the genes on the left of the genome), are not attractive since minor changes to the left most genes can cause major changes in fitness.
- Constraint 3: Not eliminated.
- Constraint 4: Eliminated, since dates booked by other events are not on the list of those available.
• Constraint 5: Eliminated, since fixed games can be added to the schedule after the algorithm completes; the dates they consume cannot be double booked (see the last point).
• Constraints 6 and 7: Eliminated (same reason as constraint 1).
• Constraint 8: Not eliminated, since it is not clear how to group games into rounds.

Unfortunately, even when ignoring constraint 8 experimentation showed that finding a feasible solution was a difficult task for a simple GA (see section 0 for a discussion of simple GAs). For example, with an eight team competition (56 games) and 64 available dates for each venue, no feasible solutions were found after a million generations (population size: 400). While we can imagine improvements by using more advanced genetic algorithms like GENITOR [Whitley, p.29] or CHC [Whitley, p.30] it seems a better encoding is required to try to remove constraints 2 and 3.

### 4.2.2 A Short Encoding

In the previous section, constraint 8 was ignored because there was no obvious way to generate game dates and then group games together into rounds. The encoding described below handles this problem by using a simple algorithm (section 4.3.1) to generate the games for each round before trying to match games to available dates at venues. It is represented pictorially below for a league of six teams:

![Diagram of encoding](attachment:encoding_diagram.png)

Each value is a team identifier in the range [0..N] for a N team competition.

The length of the genome is equal to the number of teams in the competition. No value may appear twice.

---

16 CHC stands for "Cross generational elitist selection, Heterogenous recombination and Cataclysmic mutation."
For each individual, we perform the following steps:
1. Follow the algorithm in section 4.3.1 to generate a list of games.
2. For each game generated by step 1, select the home team & venue (section 4.3.1), and then the date (section 4.3.3).
3. Assess the fitness of the individual (section 4.4).

The hard constraints listed on page 30 are handled as follows:
- Constraints 1,8: Eliminated, since games are generated from the genome using the algorithm defined in section 4.3.1.
- Constraints 2,3,4,5,6,7: Deferred to deterministic algorithm (section 4.3.3).

### 4.2.3 Discussion of Short Encoding

The encoding essentially represents a pattern (see section 2.3.1), and the GA is therefore trying to find the best available pattern. The search space is limited to $N!$ (with $N$ limited to sixteen teams for most competitions), since there are $N!$ permutations of a list of $N$ teams\(^{17}\).

There are now two questions to consider:
1. Since the search space is relatively small, should we just do an exhaustive search?
2. While the encoding ensures legal fixtures, can all possible fixtures be generated?

Examining the first question, we note that the fixture generation, is a relatively involved process, and also that while $N$ is expected to be relatively small, the search space is still increasing at a super-polynomial rate. Hence we still wish to improve the search, and in this paper we are investigating genetic algorithms. Whether the GA proves better than exhaustive search of all $N!$ possibilities is discussed in section 5.3.5.1.

To answer the second question, we first note that the list of games generated using the algorithm in 4.3.1 does not specify the home team, venue or date -

---

\(^{17}\) Note that a search space of $N!$ does not indicate a search time of $N!$, since GAs do not guarantee an optimal solution.
these are found using a rule based system; it provides a list of games within rounds. Hence the question resolves to asking whether there are fixtures that contain rounds that cannot be generated by the algorithm. We make three points:

- Enumerating all tournaments is not computationally tractable (see section 2.3.1, page 15).
- Results must be obtained in a short time frame, else we compromise the interactive abilities of the algorithm.
- An optimal result is not necessary, since the definition of optimality varies when club officials and league administrators view the results – inevitably, there will be manual alterations. Obtaining workable solutions quickly is more important; hence the need for an interactive system.

### 4.3 Scheduling Games

We have seen an encoding, and have been told that a list of games can be generated from the encoding. In this chapter, we will examine this game generation algorithm, and explore how those games are scheduled. Note that these rules are not in the objective function, but use an individual’s genetic information as a starting point.

#### 4.3.1 The Game Generation Algorithm

The game generation algorithm operates as follows:

1. List the teams according to the genetic material of the individual selected from the GA population. If there is an odd number of teams, then one ‘virtual’ team is added at the start of the list.
2. Generate a round of games: team 1 plays team N, team 2 plays team N-1, etc., where team N is the last team in the list. If the virtual team is playing, its opponent has a bye in this round.
3. Rotate teams between position 2 and position N: team 2 becomes team 3, team 3 becomes team 4,..., and team N becomes team 2.
4. Return to step 2 until the original team in position 2 returns to that position.
This produces a list of games within each round such that each team plays every other team exactly once\(^\text{18}\). These games can then be scheduled round by round.

### 4.3.2 Choosing the Home Team

Choosing the home team is implemented as a prioritized rule base. The rules are shown below; note that byes are not counted as home games.

1. If the two teams have played before in the current season, then reverse the home and away teams from the (most recent) earlier game. E.g. if North Melbourne Giants played the Brisbane Bullets at the Giant's home venue earlier in the season, then this game should be played in Brisbane.

2. Let the number of consecutive home games (counting backwards from the last game scheduled) for the first team be \(n_1\), and for the second team be \(n_2\). Further, let \(n_{\text{max}}\) be the maximum number of consecutive home games allowed.
   
   2.1. If \(n_1 > n_2\), and \(n_1 > n_{\text{max}}\), then the second team should be the home team.
   
   2.2. If \(n_2 > n_1\), and \(n_2 > n_{\text{max}}\), then the first team should be at home.
   
   2.3. If neither 2.1 nor 2.2 are satisfied, then continue to rule 3.

   This rule is required to satisfy Constraint 12 (see section 3.3).

3. If the first team is on a 'road trip', then it is set to the away team. If not, then the same rule is applied to the other team. A team is said to be on a road trip if two conditions are met:
   
   3.1. It was playing away last game.
   
   3.2. The team it played last game was the road trip partner of its opponent this game.

Note that the term 'last game' was used in place of the term 'last round'. This is important, since the last game within a round is not necessarily played before the first game the next round. This allows for a game within a round to be designated as a 'fixed date' game (see section 4.3.3).

\(\text{18}\) If teams should play each other twice or more, the list of games can simply be repeated.
4. Lastly, the team that has played the most home games so far this season is set to the away game this season. If both teams have had the same number of home games, then the first team is arbitrarily chosen to play this game at home.

4.3.3 Specifying the Game Date

The game date is calculated after the venue is chosen, and the calculation also utilizes a prioritized rule base approach. Of course, this procedure does not take place if the game is a bye.

1. Check if this game is a 'fixed date game'. Note that this check depends upon which team has been chosen as the home team - A vs. B is not the same game as B vs. A.

2. Check whether this game is a road trip game for the away team. If so, then try to make the game date as close as possible to the date of the last game that the away team played (gd_last). Each date, starting from the first day after gd_last is checked to find out whether it is available\(^{19}\). This rule is used to satisfy constraint 11 (page 30). If no available date is found after looking for a predefined maximum (presently set arbitrarily to 14 days), then we move to point 4.

3. Check the preferred weekday for availability. The date of the preferred weekday is calculated by first finding a reference date, which is the season start date plus one week for every round of games (see section 3.4), and then finding the preferred weekday closest to this date. For example, if the preferred weekday is Wednesday, and the season start date was a Tuesday, then the date calculated will be one day after the reference date. If the preferred weekday is available, then that is when the game is scheduled.

4. Check all other days of the week, starting from the closest Friday to the reference date of this round.

5. If a date can not be specified, and the game is in the first half of the season, switch the home and away teams, and check points 1 through 4 again. This

\(^{19}\) Available means that both the venue and the team are not booked for that date.
might occur if the venue was booked out for a long period of time by another event\textsuperscript{20}. This rule only applies in the first half of the season since we must ensure that each team has the same number of games during the season. If neither venue is available, the game shall be arbitrarily scheduled on the date 14 days from the start of the round, and the game is marked as invalid. This allows the problem to be brought to the attention of the league’s administration staff.

4.4 Example

To illustrate the encoding and subsequent schedule generation, we shall imagine a competition with four teams, known as ‘Blacks’, ‘Whites’, ‘Reds’ & ‘Blues’. Each of these teams is assigned a number for use in the encoding: Blacks = 0, Whites = 1, Reds = 2 and Blues = 3. Blacks and Blues are geographical close together, and it has been stipulated that they form a road trip pairing. Finally, the game between Whites and Blacks at Whites’ home venue is expected to be a game with a large public interest, and so it has been set aside for a public holiday.

4.4.1 Who plays Whom?

A chromosome might have the following genetic information:

\[
\begin{array}{c|c|c|c}
3 & 1 & 0 & 2 \\
\end{array}
\]

Using the algorithm presented in section 4.3.1 step 1, we generate the first round:

Blues vs Reds
Whites vs Blacks

In step 2, we rotate the genome from positions 2 to N:

\[
\begin{array}{c|c|c|c}
3 & 0 & 2 & 1 \\
\end{array}
\]

\textsuperscript{20} An example of this occurs when the Australian Tennis Open, which is conducted over the course of two weeks (plus preparation time) is played at the National Tennis Centre. This venue is the home of three teams in the NBL.
Which gives the next round:

Blues vs Whites
Blacks vs Reds

Repeat the rotation for the next round:

| 3 | 2 | 1 | 0 |

Hence:

Blues vs Blacks
Reds vs Whites

If we rotate again, we notice we have returned to the starting point, and so each
team has played every other team once. We can then simply replicate the
fixture if teams must play each other a multiple number of times.

4.4.2 Where do they Play?

Pick the first game in the first round, Blues vs Reds. The first rule in section
4.3.2 does not apply, since the teams have not yet played each other before this
season. Nor do the second, third or fourth rule, since neither team has had any
previous games. Hence the home team is chosen at random and the same
applies for the other game in the first round. We shall say that Blues and
Whites will play at their home venues.

In the first game of the second round, Blues and Whites have both had one
home game each, and so the home team is again chosen at random; in this case,
the Blues will play at home. In the second game, we have a more interesting
situation: Blacks and Blues are road trip partners, and the Reds team played the
Blues team at Blues home venue in the previous round. According to rule
three, Reds should play away from home again.

Blacks play Blues in the first game of the third round; Blacks have had fewer
home games than Blues so far, and so they will play at home on this occasion
(rule 4). Similarly, Whites have played more home games than Reds, and so
Reds will play at home on this occasion.
If the teams play another set of rounds (so that each team plays every other team twice), then rule one will apply in each case: the home team will be switched with the away team. For example, when Blues and Reds meet again, Reds will play at home. In this way, every team plays the same number of home games – the fixture is ‘fair’.

4.4.3 When are the Games?

We shall assume the season commences on Wednesday February 1st.

In the first round, Blues play Reds at home. Blues prefer to play their home games on Wednesdays. However, Wednesday of the first week of the season is unavailable, and so rule four in section 4.3.3 applies, and the teams play on Friday (the 3rd). Whites vs Blacks is a fixed date game (rule 1). The second round again sees Blues at home, and this time Wednesday (the 8th) is available. The other game has Reds on a road trip. Rule two attempts to make this game as close as possible to the last game, and since Blacks venue is available on the Saturday night (the 4th), that is when the game is scheduled.

In the third round, Blacks play Blues. Unfortunately, Blacks venue has been booked out by a concert series running over two weeks. Rule five then applies, and the game is switched to Blues home venue, which is available. If the teams play each other twice during the season, each team will still have the same number of home games, since Blacks will now play at home against Blues in the second half of the season. In the other game in the round, Reds play Whites, but Reds stadium is booked for the Friday and Saturday nights. Sunday, however, is free, and so the game is scheduled for that day.

4.5 Evaluating Fixtures

4.5.1 Introduction

When a fixture has been generated from the genetic encoding of an individual, it needs evaluation in terms of the objectives of the (human) scheduler. Assessing the fitness of an individual that violates hard constraints often depends on the difficulty involved in obtaining a feasible solution; if it is a formidable task, we
might expect a large number of infeasible solutions to pollute the population. In this case, we want to encourage individuals that are 'close' to feasible. However, if the problem is more one of optimization, then we want reasonably harsh penalties to prevent the possibility of poor solutions reproducing. The harshness is limited by the need to maintain genetic diversity in the population, and hence avoid premature convergence (see section 2.2.2). Note the usage of terms such as 'reasonably' in the last paragraph - there is little or no published research into mathematical methods for generating penalty/award values, largely because such methods would be overly specific to a particular problem domain. Further, the values need to be subjective since every human scheduler will have a different perspective on what is important, and what is not - the values will remain 'tweaking' control parameters.

### 4.5.2 Configuration

As mentioned in the introduction (chapter 1), the scheduling must be as configurable as possible: all rewards/penalties must be modifiable, and constraint parameters must also be user-editable where possible. Shown below is a list of the parameters a user can alter:

- The maximum number of consecutive home games.
- The penalty for scheduling too many consecutive home games.
- The maximum number of days between games on a road trip.
- The penalty for exceeding the maximum number of days between road trip games.
- The reward for scheduling each road trip.
- The reward for scheduling a team on its preferred weekday.
- The penalty for failing to schedule a game at an available venue.

These parameters are in addition to the more basic configuration data, which tend to be fixed by the problem: for example, the dates available for each venue can be considered outside the control of the user. To see an example configuration file, see Appendix A, and for more information about using the software, see Appendix D.
4.5.3 The Objective Function

The objective function is responsible for assigning a fitness value\textsuperscript{21} to each individual in the population.

Shown below is a flow chart of how the fitness function assesses a given individual. The full objective function, in C++ code, is shown in Appendix C. The ‘schedule according to team order’ action is the topic of section 4.3 (page 37) and creates a complete fixture from the genetic information of the individual that is being assessed. Note that penalties are negative values, and that the acronym 'RTP' refers to a Road Trip Partner (see glossary).

\textsuperscript{21} This value may then be scaled by the GA to maintain selective pressure.
Start

Obtain scheduling parameters and initialise fitness to zero

Schedule according to team order

For each team

For each game in this teams schedule

If game is on preferred weekday, add reward

If game is at home, and there are too many consecutive home games, add penalty

Is this game, and the next, away?

Yes

Is the opponent a RTP of the next opponent?

No

Yes

Add Road Trip reward

For each day between this and the next game, in excess of the maximum allowed between road trip games, add penalty.

All games finished?

No

Yes

All teams finished?

Yes

No

Return fitness

Figure 2: The Objective Function
4.6 Making the Scheduling Interactive

Even given a fast algorithm, there is still some work required to make the scheduling interactive in the senses defined in the introduction (page 2), and this is handled in two steps. Firstly, if the system allows fixed date games to be specified, then a league administrator can take the results of one run, fix the dates of as many games as he/she sees fit, and then run the system again. This process is described in detail in Appendix D.9 (page 81) for the system developed. Secondly, a GA can be stopped after any given generation to modify constraints (Appendix D.7, page 79).
Chapter 5: Experimental Results

5.1 Introduction

Software was implemented to test the algorithm. This software was tested in two stages, the first stage designed to trial three different GA variants: Steady State (see section 2.2.4, page 8), Incremental\textsuperscript{22} & Simple (section 2.2.1, page 5). These trials used the configuration information (and variants of the configuration values) shown in appendices A & B. Each run in the first stage had the base configuration summarised below:

- Twelve teams
- Ten venues - three teams share one venue (the National Tennis Centre).
- Two road trip pairs: Adelaide and Perth, Townsville and Brisbane. These were chosen for their geographical remoteness to the majority of NBL teams, and proximity to each other.
- One fixed date game.
- Start date December 1, 1997.
- Five days in December, two days in January and ten consecutive days in February are unavailable at every venue. This unavailability was based upon discussions with the NBL.
- Maximum consecutive home games: 3.
- Maximum number of days between road trips: 3.
- Crossover probability = 0.6 and mutation probability 0.01. These values were chosen from informal experimentation.
- The steady state GA introduced five new individuals in each reproductive cycle, replacing the five least fit individuals of the current population.

The second stage of experimentation examines the consequences of extending the number of teams. This is intended to give an indication of the applicability

\textsuperscript{22} The incremental GA is a type of Steady State GA that replaces just one individual per generation.
of the algorithm to other competitions. The GA variant with the best performance (as found in stage 1) is used for these experiments. Note that an odd number of teams (11) was also tested, with venue availability as per the NBL. As mentioned earlier (section 4.3.1, page 37), the algorithm handles this by simply inserting byes into each round. This was tested functionally, and performance results are included in section 5.4 on page 57.

5.2 Genetic Algorithm Parameters

5.2.1 Operators

While the design of the genome encoding is important in minimising 'lethal' individuals in the search space, it is equally important to select genetic operators that are compatible with the encoding. The operators chosen for experimentation are detailed here.

5.2.1.1 Mutation

The mutation operator used during the genetic search was a simple swap mutator - two positions in the same genome are chosen at random, and their positions are switched. This operator was chosen over other mutators as it preserves the feasibility of the solution; see section 2.2.6.3 (page 11).

5.2.1.2 Crossover

There are several methods used to avoid superposition\(^{23}\) of genes when exchanging genetic material between individuals, falling into two categories: permutation based crossover techniques (section 2.2.6.1) and genetic repair (section 2.2.6.2). The former category was chosen for simplicity, and PMX, OX and CX were trialled. No significant differences were detected, and so order crossover was used for subsequent experimentation.

\(^{23}\) In our case, a team appearing more than once in the genome.
5.2.1.3 Selection
The software package used for experimentation (GAlib [Wall]) provides six predefined selection schemes, including rank, roulette wheel and tournament selection. Roulette wheel selection was chosen for experimentation, after informal testing showed little difference in performance between the alternatives.

5.2.2 Fitness Scaling
Sigma truncation scaling [Goldberg] was used to maintain selective pressure on the population, since this method scales the population according to the standard deviation of fitness scores. Selective pressure is therefore the same at the start and the end of each run, despite the narrowing range of fitness scores as the population converges.

5.2.3 Replacement Schemes
When a GA that uses overlapping generations (e.g. Steady State or Incremental) is used, we have to decide which individuals in the previous generation will be removed. The experimental results removed the worst individual (incremental) or individuals (steady state) to try to maintain selective pressure.

5.2.4 Termination
Termination of the algorithm occurs when the first of the following becomes true:

- When the configurable number of generations is reached. This was set to 200 for all experiments.
- When the population has ceased improving. This is calculated as an index (i) over a configurable number of generations (n) by comparing the best individual within a generation against the best individual in the (current-n) generation. For all experiments, \(i_{\text{max}}\) was set to 0.99, and \(n\) was set to 20. This means that the best individual has improved by less than 1% over 20 generations.
5.3 Stage 1: Testing GA Variants

5.3.1 Notes

- In each run the algorithm terminated according to the convergence criteria (section 5.2.4); i.e. the maximum of 200 generations was never reached.
- Each row of each table in the tables on the following pages is the result of five runs, with each run using a different seed from the software's random number generator.
- The times were recorded with the POSIX \texttt{time} utility, with the QNX 4.22 operating system running on a Hewlett-Packard Pentium 90 MHz system.
5.3.2 Steady State GA

<table>
<thead>
<tr>
<th>Population Size</th>
<th>Average Time (sec)</th>
<th>Average Best Score</th>
<th>Best Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>16.3</td>
<td>105</td>
<td>115</td>
</tr>
<tr>
<td>20</td>
<td>33.4</td>
<td>116</td>
<td>120</td>
</tr>
<tr>
<td>50</td>
<td>102</td>
<td>121</td>
<td>126</td>
</tr>
<tr>
<td>100</td>
<td>131</td>
<td>126</td>
<td>127</td>
</tr>
<tr>
<td>200</td>
<td>305</td>
<td>133</td>
<td>134</td>
</tr>
<tr>
<td>500</td>
<td>939</td>
<td>131</td>
<td>134</td>
</tr>
</tbody>
</table>

Table 3: Results from Steady State GA

Figure 3: Fitness of best fixture vs. execution time for SS GA.

\[ y = 6.7238 \ln(x) + 90.5 \]
5.3.3 Incremental GA

<table>
<thead>
<tr>
<th>Population Size</th>
<th>Average Time (sec)</th>
<th>Average Best Score</th>
<th>Best Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>8.14</td>
<td>106</td>
<td>121</td>
</tr>
<tr>
<td>20</td>
<td>9.68</td>
<td>109</td>
<td>121</td>
</tr>
<tr>
<td>50</td>
<td>15.0</td>
<td>120</td>
<td>123</td>
</tr>
<tr>
<td>100</td>
<td>24.9</td>
<td>121</td>
<td>124</td>
</tr>
<tr>
<td>200</td>
<td>42.9</td>
<td>120</td>
<td>124</td>
</tr>
<tr>
<td>500</td>
<td>134</td>
<td>122</td>
<td>124</td>
</tr>
</tbody>
</table>

Table 4: Results from Incremental GA

Figure 4: Fitness of best fixture vs. execution time for Incremental GA

\[ y = 5.4598 \ln(x) + 99.088 \]
5.3.4 Simple GA

<table>
<thead>
<tr>
<th>Population Size</th>
<th>Average Time (sec)</th>
<th>Average Best Score</th>
<th>Best Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>42.8</td>
<td>115</td>
<td>117</td>
</tr>
<tr>
<td>20</td>
<td>55.6</td>
<td>120</td>
<td>127</td>
</tr>
<tr>
<td>50</td>
<td>218</td>
<td>127</td>
<td>128</td>
</tr>
<tr>
<td>100</td>
<td>279</td>
<td>127</td>
<td>128</td>
</tr>
<tr>
<td>200</td>
<td>593</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>500</td>
<td>1462</td>
<td>129</td>
<td>133</td>
</tr>
</tbody>
</table>

Table 5: Results from Simple GA

Figure 5: Fitness of best fixture vs. execution time for Simple GA

\[ y = 3.7052 \ln(x) + 104.48 \]
5.3.5 Discussion

5.3.5.1 Summary

- All hard constraints were met - no team played more than once on the same day, no venue was double booked, etc.
- Reasonably high scoring individuals were consistently found with very small population sizes: around 20 individuals.
- The Steady State GA provides the best performance of the three GA’s.
- The Genetic Algorithm approach is far superior to exhaustive search for a league of twelve teams. This can be seen if one notes that even the slowest GA run (simple GA with 500 individuals) assessed less than 100,000 individuals (since fewer than 200 generations were required for all population sizes). An exhaustive search would require assessment of $12!$ individuals (see section 4.2.3, page 36), a factor of 4800 more. Discussion of the quality of the results is in the next section.
- The incremental GA is the fastest GA (as expected, since only one new individual per generation needs its fitness calculated), but has lower performance than the two other algorithms.
- The highest scoring individuals in any given run were found quite quickly; only one run exceeded 60 generations (a Simple GA, with population size 10, had 73 generations).
- All GA results reached a plateau after a certain time; for example, the Steady State GA does not really improve in its performance after 360 seconds of execution time.
- The highest score obtained was 134. A discussion of what this score means is contained in the next section. This score was only obtained by the steady state GA with population sizes of 200 and 500.
- More than one individual is capable of obtaining the highest score - during testing, it was achieved 4 times by two individuals.
5.3.5.2 Discussion of Generated Fixtures

An individual with the following encoding generated the best score of 134:
11 10 5 2 7 1 9 8 3 0 6 4

These numbers refer to the team numbers as listed in the configuration file (see Appendix B). Hence the first team (number 11) refers to the Adelaide 36ers, the second team is the Hobart Tassie Devils, etc.

The fixture was examined, and no hard constraints (except one: explained below) were violated - all teams played all other teams, each team played the same number of home games, no team played another twice at home, etc (see section 3.3). The constraint violated was that some games could not be scheduled at available venues; nevertheless, the scheduler could not be considered to have failed, because ten days during February were made unavailable for all venues. The only option open to the scheduler during this period is to schedule two road trips for the preceding weekend - and this is what it did. So, in a sense, the scheduler achieved an optimal solution on this front.

To examine the fixture more closely, we refer to the score break down:

<table>
<thead>
<tr>
<th>Description</th>
<th>Occurrences</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Home Games Scheduled on Preferred Weekday</td>
<td>96</td>
<td>96</td>
</tr>
<tr>
<td>Road Trips</td>
<td>15</td>
<td>75</td>
</tr>
<tr>
<td>Too many days between road trip games</td>
<td>22</td>
<td>-22</td>
</tr>
<tr>
<td>Too many consecutive home games</td>
<td>1</td>
<td>-3</td>
</tr>
<tr>
<td>Date Unavailable</td>
<td>4</td>
<td>-12</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>134</strong></td>
<td></td>
</tr>
</tbody>
</table>

Table 6: Breakdown of highest scoring fixture

To see how this compares with an optimal value, there are 22 rounds in a season, and 6 games per round = 132 home games. Further, there are two road trip pairings (Brisbane/ Townsville & Adelaide/ Perth), giving a possible 20 road

---

24 Another individual (encoding: 7 10 3 1 9 2 8 0 4 11 6 5) achieved the same score with a different fixture. Interestingly, the number of road trips for this fixture is the same.
trips. On the surface, therefore, the performance of the system looks only average. However, the calculation above is not quite correct: while it is true that one road trip pairing allows for 10 road trips (assuming we ignore other constraints), it does not follow that two pairings allow 20 road trips. Consider the following:

- For the first pairing, the two teams must alternately play at home over the course of 20 rounds; only two rounds remain, and these must be set aside for the teams in the pairing to play each other. Hence, neither of these teams may participate in road trips visiting the other pairing - so there is now a possible 18 road trips in a season.

- When the second pairing play their road trip games against the first pairing, they are (of course) away from home for two games. This means that other teams cannot play road trips against them during these rounds, and so the possible number of road trips reduces to 16.

- However, the fixture above only achieved 15 road trips, so there is still one more road trip to account for. The table below, generated by the software before it included a penalty for unavailable dates, sheds light on this:

<table>
<thead>
<tr>
<th>Description</th>
<th>Occurrences</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Home Games Scheduled on Preferred Weekday</td>
<td>95</td>
<td>95</td>
</tr>
<tr>
<td>Road Trips</td>
<td>16</td>
<td>80</td>
</tr>
<tr>
<td>Too many days between road trip games</td>
<td>7</td>
<td>-24</td>
</tr>
<tr>
<td>Too many consecutive home games</td>
<td>1</td>
<td>-3</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>148</td>
</tr>
</tbody>
</table>

Table 7: Breakdown of score from algorithm without date available penalty

Notice that the optimum number of road trips (16) was found without the constraint of finding venues with available dates. One road trip has been lost to satisfy this constraint, because the penalty (-6) for not scheduling two games at available venues is greater than the reward (5) for the extra road trip. This is an
example of the tradeoffs that the administrator using this software must decide upon.

5.4 Stage 2: Testing Larger Competitions

In stage 2, we trial the best performed algorithm as found in stage 1 over a wider range of possible competitions. From the results of stage 1, experiments in stage 2 were undertaken with the following parameters:

- Steady State algorithm is used.
- A population size of 50 was chosen for these tests since it gave good performance in a relatively short space of time.
- Competitions with 11, 12, 14, 16 & 18 teams, playing in both double and triple round robin tournaments.
- Each competition was conducted over 18 weeks according to the venue availability supplied by the NBL.
- Venues were created to accommodate the extra teams. Each of these venues had full availability. Number of venues = \((\text{number of teams} \ - \ 1)\) for all experiments.
- Each experiment was executed 10 times.

Note that execution times should not be compared with the previous stage of experimentation, since a different computer and operating system (Pentium II 266 MHz, Windows NT 4.0) was used.
5.4.1 Discussion of Execution Time

From Figure 6 and Figure 7, we can see that the execution time of the system is almost linear with an increasing number of games to schedule. This is largely due to the fact that the number of generations until the population converged was seemingly independent of the number of games played – the algorithm
tended to converge around 40 to 50 generations. This is an indication that more experimentation is required to ascertain whether the algorithm is converging prematurely for these larger competitions.

Hence, the 18 team competition was re-tested with a population size of 500 individuals over 10 runs. This resulted in much longer execution times (by a factor of 10), and an improvement in the score of the best individual was of the order of 9%. Interestingly, the number of generations before convergence was less than for the smaller population.

<table>
<thead>
<tr>
<th>Population Size</th>
<th>50</th>
<th>500</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean Evaluation Score of Best Individual</td>
<td>213</td>
<td>231</td>
</tr>
<tr>
<td>Mean Execution Time</td>
<td>185</td>
<td>2007</td>
</tr>
<tr>
<td>Mean number of generations</td>
<td>52</td>
<td>44</td>
</tr>
</tbody>
</table>

Table 8: Results for 18 team double round robin competition with population sizes of 50 and 500.

As discussed in the literature survey, other techniques may also be used to extend the converge times, such as increasing mutation rates.

**5.4.2 Discussion of Functional Performance**

It is difficult to judge the absolute performance of the larger competitions for at least two reasons:

1. We cannot claim the input is grounded in reality for the larger competitions, since league administrators did not provide information such as venue availability (since these venues do not exist!).
2. We do not know what the (pareto-)optimal solution is, and we are unable to compare those generated to any created by league administrators.

5.5 Discussion of Interactive Scheduling

It is difficult to quantify the benefits of an interactive scheduling system; the attractiveness of the approach is based on improving the flexibility of the system, so that the user can revise their requirements as the schedule is developed (see section 2.3.5). Since this system has not been thoroughly trialled by sporting league administrators, we make no judgement in this area.

5.6 Validation

The official responsible for scheduling the Australian National Basketball League (NBL) has reviewed the approach and sample results, and compared it with the current approach. At present, a University mathematics department lecturer creates the framework of the fixture, which is then presented to the NBL for fine-tuning. This fine-tuning absorbs a large amount of the time of the NBL staff, who use a visual (i.e. wholly manual) approach, that involves displaying the games in a carefully formatted table, similar to that shown in Table 9. The NBL intend to trial the software for future fixtures.
Weeks

Table 9: Sample of NBL fixture format (portion only)

Week number

BB play NM, in Brisbane, in the first week, on Friday

'Italics indicates an away game

‘W’ indicates that the game is actually on a Wednesday

Unfortunately, it is difficult to quantify whether the fixtures produced by the system are the equal or better of those produced by hand, since there are so many variables involved. One possibility is to run the fixtures actually used by the NBL through the objective function used by the GA; however, this is not a satisfactory approach to validation, because it will not tell us whether soft constraints or preferences have been omitted. Further, the trade-offs between the variables involved may have been different at the time the 'real' fixture was created.

Nevertheless, it is possible to have a qualitative feel for the work, and NBL officials have stated their support for the approach used. The support is based on the reduction in human effort to create workable schedules. The next step, of course, is to use the system on 'live' data for the next season. This is the subject of future work.
Chapter 6: Conclusion

The goal of this thesis was to find a genetic algorithm for the problem of iteratively scheduling round robin national sporting competitions. The algorithm must be efficient enough to be implemented in a software system that will run in a short amount of time on a readily available personal computer. The research presented was grounded in reality by using the large number of constraints faced by the Australian National Basketball league. This grounding focussed the thesis on finding good fixtures quickly, rather than attempting to find an optimal fixture - requirements and their weightings are not absolute, and often change from iteration to iteration. Even more importantly, we must remember that the weightings are only subjective values.

An encoding was presented that allowed for a very small search space, thereby permitting use of very small populations and a small number of generations. The encoding takes advantage of the regular nature of scheduling sporting fixtures, by presenting an algorithmic scheduler with a starting point: an ordered list of the teams participating.

The algorithmic scheduler is responsible for generating a list of when teams play each other from the ordered list of the participating teams, and then selects the game date, using a five level rule base (described in section 4.3.3 on page 39). The venue is selected from the home team. At no time can a fixture be generated that violates hard constraints\textsuperscript{25}, even during the course of the GA run. The GA is used to select from the schedules generated by the algorithmic scheduler, by giving each schedule a score based on how well it satisfies the soft constraints. The rewards/penalties and parameters for these constraints are entirely user configurable. An important feature of the algorithm is the ability to specify

\textsuperscript{25} Unless no venues are available for an extended period.
games that are on a fixed date, since this allows league administrators to iteratively refine the schedule.

After using three different types of genetic algorithms, it was found that the steady state GA with a population size of around 500 individuals gave the best performance (in terms of evaluation score). However, a population of 50 individuals gave acceptable results in a much faster time frame, and is therefore more desirable because of the interactive nature of scheduling sporting competitions - each run of the software completes in only a few minutes on readily available PC hardware. The steady state GA was then tested with larger competitions, and it was found that the time required for the GA to converge increased at a nearly linear rate with an increasing number of games.

This paper does not claim that the algorithm outlined is the most efficient that exists today. The literature survey details two other papers ([Schaerf] & [Henz]) that use quite different approaches to solve similar problems in short time spans. However, we believe sufficient evidence has been given to show that further development is merited, and proposals for such development have been put forward (Chapter 7: p. 64). With this work in place, it is envisaged that league administrators will have a tool capable of rapidly generating and fine tuning good quality tournaments.
Chapter 7: Future Work

7.1 Additional Constraints proposed by the NBL

As is common in software engineering, presentation of a prototype often inspires potential users to generate additional ideas for new features, and allows pinpointing of under specified components. In this case, the following additional constraints on the scheduling engine were proposed:

1. 'Pencil' bookings. Venues used by the NBL are often nominally booked by other organisations - i.e. the other organisation is 'pencilled in' on that date. On occasion, the NBL can use its crowd pulling potential to secure these dates. The scheduling software may use these dates if games cannot be scheduled on available dates. It is proposed that a penalty be placed on the schedule (by the objective function) when games are scheduled for these dates.

2. There should be explicit checking to make sure teams do not have two home games on the one weekend (i.e. within 48 hours). This is because of promotional difficulties in marketing two games to the supporters within such a short period. This is a soft constraint, and so it is proposed that the objective function penalises fixtures with similar dates.

3. An extra weekday preference should be allowed for each team; i.e. a team may want to play its home games on Fridays and then Wednesdays. At present, only one preference can be configured. Two changes are required to the software for this change:
   (a) the objective function must reward schedules that have games played on the second preference weekday (of course, the reward is set by the user).
   (b) The algorithm shown in section 4.3.3 needs the additional step (after step 3) of selecting the second preference weekday.

26 Securing a date at a venue involves payment of a large non-refundable deposit.
4. Although games can currently be fixed to a venue and a date, the NBL also desire the ability to fix the home team, venue and date, and allow the away team to be selected by the software. It remains to be seen whether this requirement will be needed in practice.

7.2 Unscheduled Games

At present, some games may remain unscheduled if the venues of both teams involved are unavailable for an extended period of time (see section 4.3.3). The software could be extended to allow secondary venues for both teams; games would be automatically shifted to these venues when the primary venues are unavailable.

There are a few issues with this approach:

1. Should a penalty be applied to the fitness of the individual when games are scheduled at secondary venues? On the surface, it would seem so, but each time another reward or penalty is added to the objective function, it is made more difficult for the user to judge the trade-offs involved. Further, the execution time of the software will be increased.

2. If a penalty is used, then it probably should not be the same penalty for each venue. In some cases, playing games at the secondary venue might not be too bad if the capacity and amenities are comparable to the primary.

3. If secondary (or even tertiary) venues are introduced, the software could become even more sophisticated, by attempting to match the size of the venues to the expected crowd size. This is desirable, since the costs associated with ‘opening the doors’ of a large venue can mean a financial loss to the league or association if there is only a small crowd in attendance. That said, such decisions should perhaps be made by the administrator close to the game date - predicting crowd size before a season commences is not a trivial task. Perhaps the software could instead suggest a venue change; but the amount of information that the user needs to enter for such a suggestion may outweigh the benefit. In summary, these options need discussion with administrators.
4. Note that the NBL have stated that “teams do not have secondary venues”. Despite this, the author has seen fixtures supplied by the NBL where teams have nominated such venues when the regular stadium is not available.

7.3 **Splitting the Algorithm**

There is a need to investigate whether game dates could be better scheduled using a genetic algorithm (see section 4.3.3 to see how game dates are currently scheduled). This could be in the form of a two step process: a list of teams is still used to generate the entire list of games for each round. Further, the list of teams is still permutated by a GA. The objective function would rank the individuals according to the number of road trips generated, and the number of consecutive home games (beyond the maximum allowed).

After the GA has found the best individual, the second stage commences: each game could then take on a value from -7 to +7, representing the number of days before or after the nominal date of the round to which the game belongs. Fixed date games are excluded. Individuals would then be ranked according to the number of days between road trips, and whether home teams are scheduled to play on their preferred weekday.

A difficulty with this approach is that games from different rounds can now clash, or that a game from round $n$ may now be after a game in round $n+1$. A naïve solution is to limit the number of days to $[-3,+3]$, but then road trip games would nearly always be played mid-week. Another possibility is to simply penalize the individuals concerned, but this allows infeasible solutions to remain in the population - and this often leads to slow performance. Despite this problem, the approach is worth pursuing because it provides a logical time for the user to analyse the season before games are more rigidly scheduled, and then decide whether they want to proceed or start over.

7.4 **Usage Issues**

Future work could also consider the human interface to the system:
• Automatic analysis of schedules, to provide users with a clear understanding of why the system believes one schedule is superior to another.
• A graphical user interface, so that users can view the fixture in a format they can quickly comprehend, and so configuration of the system is much simpler.
Appendix A

SCHEDULE CONFIGURATION FILE

This configuration file allows specification of venues, teams, the season starting date, available venue dates, preferred game day for each team, road trip partners, games with fixed dates, criteria for giving awards or penalties, and the value of the rewards/penalties.

This configuration file must be named 'schedule.cfg' and put in the same directory as the scheduling executable file.

; Comments start with a semi-colon
; Blank lines are OK.
; Names with spaces or tabs must have double quotes
; Group names must be surrounded by [square brackets]
; There are four groups required: StartDate,Venues,Teams and Parameters.
; In addition, there is an optional extra group: FixedDateGames
; The groups can be in any order, and keywords within groups
; may also be in any order.

[StartDate]
; Three keywords required here: Day, Month & Year. Case is important.
; The year is in CCYY format.
Day: 01
Month: 12
Year: 1997

[Venues]
; Keywords are integers, starting from 1 and increasing.
; Each integer must be unique.
; The first entry is the venue name, surrounded by double quotes.
; Subsequent entries are optional, and relate to the number of
; dates for which the venue is "not" available.
; The date format is NMM/[CC]YY:d1,d2,d3...,dn, where N starts
; a new month, MM is an integer specifying which month it is (Jan=1)
; The slash is optional, and may be forward or backward.
; CC is an optional century specification, and d1,d2...dn are days
; of the month that are not available. Consecutive unavailable
; days may be specified by replacing the comma with a dash, eg 11-16
; indicates that days 11 through to 16 inclusive are unavailable.
; If the century specification is not present, then the century will
; be 19 if the year is in the nineties, and 20 if the year is in the
; range [00,89].
; If no dates are specified, then it is assumed all dates are available.
1: "Melbourne Tennis Centre",N12/97:2,12,20,25,26,N01/98:1,25,N02/98:10-20
2: "Brisbane Arena",N12/97:2-12,20,25,26,N01/98:1,25,N02/98:10-20
3: "Illawarra Snake Pit",N12/97:2,12,20,25,26,N01/98:1,25,N02/98:10-20
4: "Canberra Arena",N12/97:2,12,20,25,26,N01/98:1,25,N02/98:10-20
5: "Sydney Entertainment Centre",N12/97:2,12,20,25,26,N01/98:1,25,N02/98:10-20
7: "Newcastle Arena", N12/97: 2, 12, 20, 25, 26, N01/98: 1, 25, N02/98: 10-20
8: "Hobart Arena", N12/97: 2, 12, 20, 25, 26, N01/98: 1, 25, N02/98: 10-20

[Teams]
; Keywords are integers, starting from 1 and increasing.
; Each integer must be unique.

; The format of the entries is: "team name", venue ID, day of week, road trip
; partner id.
; The road trip partner id is optional.

; The team name can be any combination of characters and any length,
; and may include tabs or spaces provided it is surrounded by double quotes.

; The venue ID is an integer that must correspond to one of the keywords in
; the venue group. This specifies the home venue of the team.

; The day of week is one of: Sun, Mon, Tue, Wed, Thu, Fri, Sat. Case is important.
; This specifies the preferred day for games when the team is playing at home.
; Note that whether the games are actually scheduled on this weekday depends
; on venue availability.

; The road trip partner id is an integer that must correspond to one of
; the other teams. This causes the schedule to try and schedule successive
; games at the home venues of these teams. For example, if teams 3 & 4
; were partnered together, then when team 1 plays team 3 at team 3's home
; venue, it should play team 4 at team 4's venue the next round.
; This allows minimisation of travel costs.

1: "North Melbourne Giants", 1, Fri
2: "Melbourne Tigers", 1, Fri
3: "South East Melbourne Magic", 1, Fri
4: "Brisbane Bullets", 2, Fri, 8
5: "Illawarra Hawks", 3, Fri
6: "Canberra Cannons", 4, Fri
7: "Sydney Kings", 5, Fri
8: "Townsville Suns", 6, Sat, 4
9: "Newcastle Falcons", 7, Fri
10: "Hobart Tassie Devils", 8, Fri
11: "Adelaide 36ers", 9, Fri, 12
12: "Perth Wildcats", 10, Fri, 11

[FixedDateGames]
; Here games that must be played on a certain date are set.
; The format is: id: team id, team id, venue id, day of month, month, year
; team id is an integer, and must correspond to one of the ids in the
; Team group. The first team id is the home team.
; The year field is in the format: CCYY.
; The date set must be after the starting date.
;
; Note that if the date of a game is fixed, then this
; overrides the venue availability

1: 1,2,1,23,12,1997

[Parameters]
; This section allows setting of optional scheduling parameters,
; such as the maximum number of consecutive home games, and rewards
; and penalties to be used when searching for the best possible fixture.
; Note that penalties are negative, rewards positive.
MAX_CONSECUTIVE_HOME_GAMES: 3
MAX_DAYS_BWROAD_TRIP_GAMES: 3
MAX_DAYS_BW_RT_GAMES_PENALTY: -1
ROAD_TRIP_REWARD: 5
CONSECUTIVE_HOME_GAME_PENALTY: -3
PREFERRED_WEEKDAY_REWARD: 1
DATE_UNAVAILABLE_PENALTY: -3
Appendix B

GENETIC ALGORITHM CONFIGURATION

This configuration file allows specification of GA parameters, such as population size, crossover and mutation probabilities, and convergence limits. This configuration file must be called 'ga.cfg', and placed in the same directory as the executable.

score_frequency 1
flush_frequency 1
score_filename ga.dat
population_size 500
number_of_generations 200 [A maximum; used for termination]
mutation_probability 0.01
crossover_probability 0.6
generations_to_convergence 20
replacement_number 4 [Used for Steady State GA only]
record_diversity 1
THE OBJECTIVE FUNCTION

Shown below is the objective function (in the C++ programming language). It uses the GALib class library and the Standard Template Library (STL).

```cpp
float ScheduleAnalyzer(GAGenome &c) {
    // cast the genome to the type of genome we are actually using,
    // and cast the user data to a 'League' object ('cos that's what
    // it is!)
    GAL1DArrayAlleleGenome<int> &genome = (GAL1DArrayAlleleGenome<int> &c);
    SchedulingInfo *schedulingInfo = (SchedulingInfo *)c.userData();
    assert(schedulingInfo);
    League *l = schedulingInfo->l;
    Params *p = schedulingInfo->params;
    assert(l);
    assert(p);
    assert(l->NumTeams() == genome.size());
    SetLeagueTeamOrder(*l, genome);
    Schedule schedule(*l,*p,schedulingInfo->fixedDateGames);
    
    const int roadTripReward = p->RoadTripReward();
    const int maxConsecutiveHomeGames = p->MaxConsecutiveHomeGames();
    
    // (a) The number of road trips. A road trip is defined
    // prior to scheduling by pairing teams together.
    // These paired teams should then be played in successive weeks
    // by travelling teams.
    // (b) Whether the number of consecutive home games for a team
    // exceeds the maximum allowed.
    // (c) Whether fixed date games are played within +/- 6 days of the
    // round they belong within.
    // (d) Whether road trips are played close together.
    return ...
```

Appendix C

THE OBJECTIVE FUNCTION
const int consecutiveHomeGamePenalty = p->ConsecutiveHomeGamePenalty(); // Per game
const int preferredWeekdayReward = p->PreferredWeekdayReward();
const int maxDaysBwRoadTripGames = p->MaxDaysBwRoadTripGames();

float fitness=0;
for (int i=0;i<l->NumTeams();i++)
{

    Team &thisTeam = l->GetTeam(i);
    const ListGames &fixture = thisTeam.Fixture();
    int consecutiveHomeGames=0;
    for (ListGames::const_iterator iter = fixture.begin();
        iter!=fixture.end();
        ++iter)
    {
        if ((*iter).HomeTeam() == thisTeam)
        {
            // -----------------------------------
            // Check game is on preferred weekday
            if ((*iter).GameDate().Weekday() == thisTeam.PreferredGameDay())
                fitness += preferredWeekdayReward;
            // -----------------------------------
            // Check game date is actually available.
            if (!(*iter).DateUnavailable())
                fitness += dateUnavailablePenalty;
            // -----------------------------
            // Count consecutive home games
            if (maxConsecutiveHomeGames > 0 &&
                ++consecutiveHomeGames > maxConsecutiveHomeGames)
                fitness += consecutiveHomeGamePenalty;
        }
        else
            consecutiveHomeGames = 0;
        
        // -----------------------------
        // Count road trips.
// If we are playing away this week and next week...
if ((iter+1) != fixture.end() &&
    (*iter).AwayTeam() == thisTeam &&
    (*iter+1).AwayTeam() == thisTeam)
{
    // ...and if the home team this week has a road trip partner
    // who just happens to be our opponent next week...
    const Team *partner = (*iter).HomeTeam().RoadTripPartner();
    if (partner && *partner == (*(iter+1)).HomeEm())
    {
        // ...then we have a road trip!

        // Note that this takes into account fixed date games.
        // Fixed date games have the potential to throw road trips
        // out in a less well designed system than this one,
        // since successive games may not be in chronological order.
        // However, the Team::Fixture() function makes sure
        // everything is sorted in chronological order first.
        fitness += roadTripReward;

        int numDaysBwGames =
            Diff({*(iter+1)}.GameDate(),{iter}.GameDate());
        if (maxDaysBwRoadTripGames > 0 &&
            numDaysBwGames > maxDaysBwRoadTripGames)
            fitness += (numDaysBwGames-maxDaysBwRoadTripGames) * p->TooManyDaysBwRTGamesPenalty();
    }
}

// Clear all bookings
l->Reset();
return fitness;
D.1 Introduction

This section describes the software that was developed or used to perform service functions for the algorithm, and also includes a section on using the software (especially configuration). It has been included to give the reader a more complete understanding of the work involved.

D.2 GAlib

A number of genetic algorithm libraries are freely available on the Internet. These were evaluated according to the following criteria:
1. Compilable on a range of platforms (especially the development platform, QNX). This meant that ANSI C or C++ was required, since these are the only languages commercially supported on the QNX operating system. C++ was also preferred for two other reasons: languages such as Lisp were not ported to QNX, and are so are unused on that platform; and the author’s own familiarity with C++.
2. Supported a wide range of genetic operators and genetic algorithms.
3. Allows runtime modification of genetic algorithm parameters (crossover probabilities etc) and objective function rewards/penalties. This is crucial to the interactive requirements of this project.

GAlib [Wall] is an object oriented C++ library, and has a range of genetic algorithm (e.g. GASteadyStateGA, GAIncrementalGA,...) and genome (e.g. GA1DArraryAlleleGenome) classes. Each genetic algorithm can use any of the genome classes, and each genome defines a number of operators (e.g. single
point crossover, PMX etc). Further, the library can be customised or extended with relative ease.

**D.3 Software Design**

An object-oriented design was used in an attempt to divide the problem into smaller, more manageable sub-projects, thereby allowing flexibility for future work [Booch].

The key classes were relatively easy to identify through an analysis of the domain and are shown in the table below. The table does not show those classes implemented in the GAlib package, or those provided by Standard C++ (String & STL container and iterator classes).
<table>
<thead>
<tr>
<th>Class Name</th>
<th>Attributes</th>
<th>Operations</th>
</tr>
</thead>
</table>
| League     | list of teams  
             | list of venues                               | Add team  
             | Add venue  
             | Reorder teams.  
             | Get num of games  
             | Get num of rounds |
| Team       | Name  
             | road trip partner\(^{26}\)  
             | preferred game day  
             | home venue  
             | list of games scheduled | Clear all bookings  
             | Make home booking  
             | Make away booking  
             | Check date clash  
             | Order\(^{29}\) and return game list |
| Venue      | Name  
             | List of originally available dates  
             | List of booked\(^{30}\) dates | Clear all bookings  
             | Make booking\(^{31}\)  
             | Check date clash |
| Date       | Starting date (shared between all  
             | Date objects)  
             | Actual date (from start date) | Diff between two dates  
             | Get weekday. |
| Round      | A list of games  
             | Num of games (shared) | |
| Schedule   | List of rounds | Pick Home Team  
             | Check if game date is fixed  
             | Check if on road trip  
             | Set a game date |
| Game       | Date\(^{28}\)  
             | Venue  
             | Home team  
             | Away team | |
| Params     | A keyword to value mapping. |

\(^{27}\) The majority of const functions (i.e. those that do not change the state of the object) are not mentioned.

\(^{28}\) May be unspecified.

\(^{29}\) Ordering of games is by date, dates at the start of the season first.

\(^{30}\) By the league.

\(^{31}\) Bookings can be made even if the date will clash with an existing booking. The check date clash operation can be used to prevent this, if desired. This applies to both the Venue and Team classes.
**D.4 Operation**

The flow chart below shows the basic operation of the genetic algorithm part of the system:

![Flow Chart: GA Operation](image_url)

*Figure 8: GA Operation*
D.5 Software Usage

Using the software is a three step procedure: firstly, create or modify the configuration files; secondly, start the program, and watch the score of the best individual improve; and finally, analyse the fixture produced, to see if it fits your requirements.

D.6 Configuration Files

There are two configuration files: schedule.cfg and ga.cfg. Examples of both are shown and explained in Appendices A & B, respectively.

D.7 Interaction

After each generation, the generation number, best individual, best score, convergence and diversity is printed. The user can pause the algorithm by hitting a key at any time. When the algorithm is paused, the user is presented with a menu of options, allowing:

- Access to further information about the schedule (see section D.8).
- Addition, removal or listing of fixed date games.
- Modification of constraint values.
- Modification of rewards and penalties.

An example run is shown below. The first line refers to the season starting date.

Start date Mon Dec 1 00:00:00 1997

The ga generated: 4 3 5 10 8 6 9 7 0 1 11 2  gen 1 convergence 0 diversity 0.872368 best score 101
The ga generated: 4 3 5 10 8 6 9 7 0 1 11 2  gen 2 convergence 0 diversity 0.845175 best score 101
The ga generated: 4 3 5 10 8 6 9 7 0 1 11 2  gen 3 convergence 0 diversity 0.845614 best score 101
The ga generated: 4 3 5 10 8 6 9 7 0 1 11 2  gen 4 convergence 0 diversity 0.695175 best score 101
[Key pressed here]

1. Dump the fixture to fixture.txt
2. Dump the road trips to road.txt
3. Change search parameters
Any other key to continue optimisation

2 [Road trip listing dumped for user analysis]

Road trips are printed in roadtrips.txt
1. Dump the fixture to fixture.txt
2. Dump the road trips to road.txt
3. Change search parameters
Any other key to continue optimisation

3 [User pressed '3' to change search parameters]

Press r to change rewards
Press f to change fixed date games

r [User pressed 'r' to change rewards]
1. Max consecutive home games: 3
2. Max days between games on a road trip: 3
3. Penalty for too many days between road trip games: -1
4. Reward for scheduling road trips: 5
5. Penalty for scheduling too many consecutive home games: -3
6. Reward for scheduling on a preferred weekday: 1
7. Penalty for scheduling dates on unavailable days: -3

Enter a number from 1 to 7

4

Enter new integer value (then press enter key): 20

[User has now modified road trip reward to try to enhance possibility of extra road trips being generated]

1. Dump the fixture to fixture.txt
2. Dump the road trips to road.txt
3. Change search parameters
Any other key to continue optimisation

[Optimisation continued]

The ga generated: 10 4 9 5 7 6 0 3 8 2 11 1  gen 5 convergence 0 diversity 0.526754 best score 244

[Generations 6 to 33 not shown]

The ga generated: 0 5 2 10 8 6 4 3 7 9 11 1  gen 34 convergence 1 diversity 0.0166667 best score 317

Road trips are printed in roadtrips.txt

D.8 Fixture Analysis

For each schedule generated, 2 + N files (where N is the number of teams) are created:

- Fixture.txt: a listing of all the games for the season. Each game specifies the weekday, date, venue and teams. The first team mentioned is the home team (see section E.2 for an example).
• Roadtrips.txt: a list of the first game for each of the road trips in the season (see section E.1 for an example).
• A file for each team, listing all the games for that team.

These files allow the user to count the number of road trips, find out how often home teams have been scheduled to play on their preferred weekday, etc.

D.9 Progressive Refinement

The system allows the output file (Fixture.txt) to be piped back into the configuration file (schedule.cfg) in the form of fixed date games. This facilitates support for the iterative process that human schedulers often go through - generate a fixture, fix a portion of it, and reschedule the remainder.

It works as follows.

1. Generate a fixture.
2. The user examines the fixture, and deletes games from the output file that he or she wants rescheduled.
3. The user then asks the software, via a command line argument, to read in the output file, and convert each game contained therein to a fixed date game in the configuration file.
4. Return to point 1 until the entire schedule is generated.

Of course, this could be done by hand, but with a typical season having in excess of 100 games, this would be a tedious task indeed.
Appendix E

SAMPLE OUTPUT FILES

E.1 Road Trips

Roadtrip.txt lists all the road trips in a given season. Note that only the first game in each road trip is mentioned, and that the list is ordered by team, and then by date. Games labelled with an asterisk could not be scheduled because no venues were available.

Road trips are
===============
Saturday January 24
 Townsville Arena: Townsville Suns vs Adelaide 36ers

Friday April 03
 Perth Arena: Perth Wildcats vs Townsville Suns

Saturday December 13
 Perth Arena: Perth Wildcats vs Newcastle Falcons

Saturday April 18
 Townsville Arena: Townsville Suns vs Newcastle Falcons

Friday April 10
 Perth Arena: Perth Wildcats vs Brisbane Bullets

Saturday February 07
 Townsville Arena: Townsville Suns vs North Melbourne Giants

Friday March 06
 Perth Arena: Perth Wildcats vs North Melbourne Giants

Friday January 30
 Perth Arena: Perth Wildcats vs Melbourne Tigers

Saturday March 21
 Townsville Arena: Townsville Suns vs Melbourne Tigers

Saturday December 27
 Perth Arena: Perth Wildcats vs Illawarra Hawks

Thursday February 19
 * Townsville Arena: Townsville Suns vs Illawarra Hawks

Friday February 06
 Perth Arena: Perth Wildcats vs Sydney Kings

Saturday March 28
Townsville Arena: Townsville Suns vs Sydney Kings
Saturday December 06
Townsville Arena: Townsville Suns vs South East Melbourne Magic
Friday March 20
Perth Arena: Perth Wildcats vs South East Melbourne Magic
Saturday January 17
Townsville Arena: Townsville Suns vs Perth Wildcats
Friday January 09
Perth Arena: Perth Wildcats vs Canberra Cannons

End of road trip listing
E.2 Complete Fixture

Fixture.txt is a complete listing of all games for the season. It is discussed in section 5.3.5.2. Games that could not be scheduled (because no venues were available) are labelled with an asterisk.

**Round 1**
Friday December 05
Hobart Arena: Hobart Tassie Devils vs Canberra Cannons

Friday December 05
Adelaide Arena: Adelaide 36ers vs Perth Wildcats

Saturday December 06
Townsville Arena: Townsville Suns vs South East Melbourne Magic

Friday December 05
Newcastle Arena: Newcastle Falcons vs Sydney Kings

Friday December 05
Illawarra Snake Pit: Illawarra Hawks vs Brisbane Bullets

Friday December 05
Melbourne Tennis Centre: North Melbourne Giants vs Melbourne Tigers

**Round 2**
Saturday December 13
Hobart Arena: Hobart Tassie Devils vs Adelaide 36ers

Saturday December 13
Canberra Arena: Canberra Cannons vs Townsville Suns

Saturday December 13
Perth Arena: Perth Wildcats vs Newcastle Falcons

Saturday December 13
Brisbane Arena: Brisbane Bullets vs South East Melbourne Magic

Saturday December 13
Sydney Entertainment Centre: Sydney Kings vs North Melbourne Giants

Saturday December 13
Melbourne Tennis Centre: Melbourne Tigers vs Illawarra Hawks

**Round 3**
Friday December 19
Townsville Arena: Townsville Suns vs Hobart Tassie Devils

Sunday December 14
Adelaide Arena: Adelaide 36ers vs Newcastle Falcons

Friday December 19
Brisbane Arena: Brisbane Bullets vs Canberra Cannons
Friday December 19
Melbourne Tennis Centre: North Melbourne Giants vs Perth Wildcats

Sunday December 21
Melbourne Tennis Centre: South East Melbourne Magic vs Melbourne Tigers

Friday December 19
Illawarra Snake Pit: Illawarra Hawks vs Sydney Kings

**Round 4**
Saturday December 27
Newcastle Arena: Newcastle Falcons vs Hobart Tassie Devils

Saturday December 27
Brisbane Arena: Brisbane Bullets vs Townsville Suns

Saturday December 27
Melbourne Tennis Centre: North Melbourne Giants vs Adelaide 36ers

Sunday December 28
Melbourne Tennis Centre: Melbourne Tigers vs Canberra Cannons

Saturday December 27
Perth Arena: Perth Wildcats vs Illawarra Hawks

Saturday December 27
Sydney Entertainment Centre: Sydney Kings vs South East Melbourne Magic

**Round 5**
Friday January 02
Hobart Arena: Hobart Tassie Devils vs Brisbane Bullets

Friday January 02
Newcastle Arena: Newcastle Falcons vs North Melbourne Giants

Friday January 02
Melbourne Tennis Centre: Melbourne Tigers vs Townsville Suns

Sunday December 28
Adelaide Arena: Adelaide 36ers vs Illawarra Hawks

Friday January 02
Canberra Arena: Canberra Cannons vs Sydney Kings

Saturday January 03
Melbourne Tennis Centre: South East Melbourne Magic vs Perth Wildcats

**Round 6**
Friday January 09
Hobart Arena: Hobart Tassie Devils vs North Melbourne Giants

Friday January 09
Melbourne Tennis Centre: Melbourne Tigers vs Brisbane Bullets
Friday January 09
Illawarra Snake Pit: Illawarra Hawks vs Newcastle Falcons

Friday January 09
Sydney Entertainment Centre: Sydney Kings vs Townsville Suns

Saturday January 10
Melbourne Tennis Centre: South East Melbourne Magic vs Adelaide 36ers

Friday January 09
Perth Arena: Perth Wildcats vs Canberra Cannons
Round 7
Friday January 16
Hobart Arena: Hobart Tassie Devils vs Melbourne Tigers

Friday January 16
Illawarra Snake Pit: Illawarra Hawks vs North Melbourne Giants

Friday January 16
Sydney Entertainment Centre: Sydney Kings vs Brisbane Bullets

Friday January 16
Melbourne Tennis Centre: South East Melbourne Magic vs Newcastle Falcons

Saturday January 17
Townsville Arena: Townsville Suns vs Perth Wildcats

Sunday January 11
Adelaide Arena: Adelaide 36ers vs Canberra Cannons

Round 8
Friday January 23
Illawarra Snake Pit: Illawarra Hawks vs Hobart Tassie Devils

Friday January 23
Sydney Entertainment Centre: Sydney Kings vs Melbourne Tigers

Friday January 23
Melbourne Tennis Centre: North Melbourne Giants vs South East Melbourne Magic

Sunday January 18
Brisbane Arena: Brisbane Bullets vs Perth Wildcats

Friday January 23
Canberra Arena: Canberra Cannons vs Newcastle Falcons

Saturday January 24
Townsville Arena: Townsville Suns vs Adelaide 36ers

Round 9
Friday January 30
Hobart Arena: Hobart Tassie Devils vs Sydney Kings

Friday January 30
Melbourne Tennis Centre: South East Melbourne Magic vs Illawarra Hawks

Friday January 30
Perth Arena: Perth Wildcats vs Melbourne Tigers

Friday January 30
Canberra Arena: Canberra Cannons vs North Melbourne Giants

Monday January 26
Brisbane Arena: Brisbane Bullets vs Adelaide 36ers

Friday January 30
Newcastle Arena: Newcastle Falcons vs Townsville Suns
Round 10

Friday February 06
Melbourne Tennis Centre: South East Melbourne Magic vs Hobart Tassie Devils

Friday February 06
Perth Arena: Perth Wildcats vs Sydney Kings

Friday February 06
Canberra Arena: Canberra Cannons vs Illawarra Hawks

Saturday January 31
Adelaide Arena: Adelaide 36ers vs Melbourne Tigers

Saturday February 07
Townsville Arena: Townsville Suns vs North Melbourne Giants

Friday February 06
Newcastle Arena: Newcastle Falcons vs Brisbane Bullets

Round 11

Thursday February 19
* Perth Arena: Perth Wildcats vs Hobart Tassie Devils

Thursday February 19
* Melbourne Tennis Centre: South East Melbourne Magic vs Canberra Cannons

Saturday February 07
Adelaide Arena: Adelaide 36ers vs Sydney Kings

Thursday February 19
* Townsville Arena: Townsville Suns vs Illawarra Hawks

Thursday February 19
* Melbourne Tennis Centre: Melbourne Tigers vs Newcastle Falcons

Sunday February 08
Brisbane Arena: Brisbane Bullets vs North Melbourne Giants

Round 12

Saturday February 21
Canberra Arena: Canberra Cannons vs Hobart Tassie Devils

Saturday February 21
Perth Arena: Perth Wildcats vs Adelaide 36ers

Saturday February 21
Melbourne Tennis Centre: South East Melbourne Magic vs Townsville Suns

Saturday February 21
Sydney Entertainment Centre: Sydney Kings vs Newcastle Falcons

Saturday February 21
Brisbane Arena: Brisbane Bullets vs Illawarra Hawks

Sunday February 22
Round 13
Friday February 27
Adelaide Arena: Adelaide 36ers vs Hobart Tassie Devils

Saturday February 28
Townsville Arena: Townsville Suns vs Canberra Cannons

Friday February 27
Newcastle Arena: Newcastle Falcons vs Perth Wildcats

Friday February 27
Melbourne Tennis Centre: South East Melbourne Magic vs Brisbane Bullets

Saturday February 28
Melbourne Tennis Centre: North Melbourne Giants vs Sydney Kings

Friday February 27
Illawarra Snake Pit: Illawarra Hawks vs Melbourne Tigers

Round 14
Friday March 06
Hobart Arena: Hobart Tassie Devils vs Townsville Suns

Friday March 06
Newcastle Arena: Newcastle Falcons vs Adelaide 36ers

Friday March 06
Canberra Arena: Canberra Cannons vs Brisbane Bullets

Friday March 06
Perth Arena: Perth Wildcats vs North Melbourne Giants

Friday March 06
Melbourne Tennis Centre: Melbourne Tigers vs South East Melbourne Magic

Friday March 06
Sydney Entertainment Centre: Sydney Kings vs Illawarra Hawks

Round 15
Friday March 13
Hobart Arena: Hobart Tassie Devils vs Newcastle Falcons

Saturday March 14
Townsville Arena: Townsville Suns vs Brisbane Bullets

Friday March 13
Adelaide Arena: Adelaide 36ers vs North Melbourne Giants

Friday March 13
Canberra Arena: Canberra Cannons vs Melbourne Tigers

Friday March 13
Illawarra Snake Pit: Illawarra Hawks vs Perth Wildcats

Friday March 13
Round 16
Friday March 20
Brisbane Arena: Brisbane Bullets vs Hobart Tassie Devils

Friday March 20
Melbourne Tennis Centre: North Melbourne Giants vs Newcastle Falcons

Saturday March 21
Townsville Arena: Townsville Suns vs Melbourne Tigers

Friday March 20
Illawarra Snake Pit: Illawarra Hawks vs Adelaide 36ers

Friday March 20
Sydney Entertainment Centre: Sydney Kings vs Canberra Cannons

Friday March 20
Perth Arena: Perth Wildcats vs South East Melbourne Magic

Round 17
Friday March 27
Melbourne Tennis Centre: North Melbourne Giants vs Hobart Tassie Devils

Friday March 27
Brisbane Arena: Brisbane Bullets vs Melbourne Tigers

Friday March 27
Newcastle Arena: Newcastle Falcons vs Illawarra Hawks

Saturday March 28
Townsville Arena: Townsville Suns vs Sydney Kings

Friday March 27
Adelaide Arena: Adelaide 36ers vs South East Melbourne Magic

Friday March 27
Canberra Arena: Canberra Cannons vs Perth Wildcats

Round 18
Friday April 03
Melbourne Tennis Centre: Melbourne Tigers vs Hobart Tassie Devils

Saturday April 04
Melbourne Tennis Centre: North Melbourne Giants vs Illawarra Hawks

Friday April 03
Brisbane Arena: Brisbane Bullets vs Sydney Kings

Friday April 03
Newcastle Arena: Newcastle Falcons vs South East Melbourne Magic

Friday April 03
Perth Arena: Perth Wildcats vs Townsville Suns

Friday April 03
Canberra Arena: Canberra Cannons vs Adelaide 36ers
Round 19  
Friday April 10  
Hobart Arena: Hobart Tassie Devils vs Illawarra Hawks  

Friday April 10  
Melbourne Tennis Centre: Melbourne Tigers vs Sydney Kings  

Saturday April 11  
Melbourne Tennis Centre: South East Melbourne Magic vs North Melbourne Giants  

Friday April 10  
Perth Arena: Perth Wildcats vs Brisbane Bullets  

Friday April 10  
Newcastle Arena: Newcastle Falcons vs Canberra Cannons  

Friday April 10  
Adelaide Arena: Adelaide 36ers vs Townsville Suns  

Round 20  
Friday April 17  
Sydney Entertainment Centre: Sydney Kings vs Hobart Tassie Devils  

Friday April 17  
Illawarra Snake Pit: Illawarra Hawks vs South East Melbourne Magic  

Friday April 17  
Melbourne Tennis Centre: Melbourne Tigers vs Perth Wildcats  

Saturday April 18  
Melbourne Tennis Centre: North Melbourne Giants vs Canberra Cannons  

Friday April 17  
Adelaide Arena: Adelaide 36ers vs Brisbane Bullets  

Saturday April 18  
Townsville Arena: Townsville Suns vs Newcastle Falcons  

Round 21  
Friday April 24  
Hobart Arena: Hobart Tassie Devils vs South East Melbourne Magic  

Friday April 24  
Sydney Entertainment Centre: Sydney Kings vs Perth Wildcats  

Friday April 24  
Illawarra Snake Pit: Illawarra Hawks vs Canberra Cannons  

Friday April 24  
Melbourne Tennis Centre: Melbourne Tigers vs Adelaide 36ers  

Saturday April 25  
Melbourne Tennis Centre: North Melbourne Giants vs Townsville Suns  

Friday April 24
Brisbane Arena: Brisbane Bullets vs Newcastle Falcons
Round 22
Friday May 01
Hobart Arena: Hobart Tassie Devils vs Perth Wildcats

Friday May 01
Canberra Arena: Canberra Cannons vs South East Melbourne Magic

Friday May 01
Sydney Entertainment Centre: Sydney Kings vs Adelaide 36ers

Friday May 01
Illawarra Snake Pit: Illawarra Hawks vs Townsville Suns

Friday May 01
Newcastle Arena: Newcastle Falcons vs Melbourne Tigers

Friday May 01
Melbourne Tennis Centre: North Melbourne Giants vs Brisbane Bullets
bibliography


Cavicchio, D. J.: A adaptive Search using Simulated Evolution. As reported in [Goldberg], page 96.


Husbands, Philip: Genetic Algorithms for Scheduling. AISB Quarterly, No. 89.

Jones, Terry. Crossover, Macromutation and Population Based Search. Insert location and date: look in Santa Fe (www.santafe.edu)


McWorter, Prof. William A.: Personal Communication.


Index

CHC, 7, 8, 32
Configuration, 40, 65, 75
Crossover, x, 5, 6, 10, 45, 89
Diversity, 6
encoding, iii, 15, 16, 17, 19, 26, 30, 31, 32, 39, 45, 52, 63
flow charts, 74
GAlib, ix, 45, 71, 90
GENITOR, 8, 32
Hybrid, 10
JSSP, x, 18, 19
Mutation, x, 5, 10, 45
objective function, 5, 9, 16, 17, 18, 41, 61, 68, 71
Planning Windows, 2
Preactor, 2
Schedule Wizard for Windows, 25
school timetabling, 15, 17
Selection
   Rank based, 9
   Tournament, 9
Simple Genetic Algorithms, 5
speciation, 6
Team Sports Scheduling System, 25
Winterface, 2

Page 101