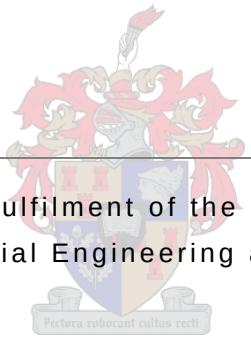
A Population-Based Approach to Sequential Ordering Problems

Carel Aäron Anthonissen



Thesis presented in partial fulfilment of the requirements for the degree of Master of Science in Industrial Engineering at Stellenbosch University.

Study Leader: James Bekker

March 2007

DECLARATION

I, the undersigned, hereby declare that the work contained in this thesis is my own original work and that I have not previously in its entirety or in part submitted it at any university for a degree.

Signature: L A Anthonissen Date: 05 March 2007

ABSTRACT

This project was initiated to develop a new and novel approach to address complex sequencing problems, in particular, an alternative method was developed to find solutions to the sequential ordering problem.

The sequential ordering problem is concerned with the arrangement of a number of elements in a sequence that respects a number of precedence constraints and results in the lowest overall cost. A precedence constraint requires that some element will occur before another in the solution sequence, and the cost of a solution is determined by summing the independent individual costs that are incurred when progressing from one specific element in the solution sequence to another.

Instances of this problem are regularly found in the practice of industrial engineering in problems such as the routing of a delivery vehicle, the scheduling of jobs on a machine and the preparation of project plans with limited resources.

The sequential ordering problem is known to be complex in the sense that as the size of problem instances increases, the best-known time required to find a guaranteed optimal solution increases exponentially.

The objective of the work in this thesis is to develop a solution that will generate an optimal or near optimal solution to instances of the sequential ordering problem within a practically acceptable time period.

The new method operates by using two interdependent tiers. The first, a so-called population-based tier, maintains knowledge about the solutions space through a diverse set of candidate solutions. This tier seeks to improve the diversity of its population and thereby to identify new and promising regions of the search space. In this project, genetic algorithms and particle swarm optimisation was used as population based methods for this tier. The second, a so-called local optimisation tier, performs an exploitative function by optimising the solutions identified in the population-based tier through "greedy" neighbourhood search methods. Such a method repeatedly improves the candidate solution through small changes to the solution sequence and continues until it is unable to improve the solution any further.

The proposed two-tiered architecture is very similar to memetic algorithms, a metaheuristic method that also combines population-based and neighbourhood search methods.

Two sets of objectives were identified for the proposed method. Performance objectives require that the method needs to find optimal or near optimal solutions within practically achievable time periods. The performance of the proposed method against this objective is determined by either measuring the time required to achieve an optimal solution, or alternatively by measuring the total cost of the best solution in a limited time trial and comparing the result against benchmark methods in solving the sequential ordering problem. In this project, a genetic algorithm using a so-called "edge assembly" crossover procedure and a simulated annealing heuristic were used as benchmark methods.

The research objective was to separate the exploitative and exploratory behaviour into the two tiers and to determine whether or not a two-tiered solver could achieve better performance than the two tiers do on their own. The same measurements that were used for the performance objective were used to compare the performance of each solver to that of its component tiers.

Against these objectives, the following operating principles were specified for the two-tiered solver architecture:

- local optima have to be identified and populated,
- population diversity has to be promoted,
- the average population performance has to be improved, and
- functional overlap between tiers has to be minimised.

To support these operating principles, different roles were assigned to the different tiers. The following roles were assigned to the population-based tier:

- to maintain knowledge from one iteration to the next,
- to fulfil the exploratory function of the solver, and
- to apply problem independent techniques, *i.e.* it avoids exploitation of problem specific knowledge.

The following roles were assigned to the local search tier:

- to exploit newly generated solutions to identify local optima,
- to define predictable and stable basins of attraction for local optima, and
- to use problem specific knowledge to improve the solver's performance.

For all the implementations of the two-tiered solvers and their component tiers, constraints were addressed using the following methodology:

For each violated constraint, the degree to which this constraint was violated is quantified. The degree of constraint violation for a solution is defined as the sum of all the degrees of violation of

all the violated constraints. A solution with a lower degree of constraint violation is always considered superior to one with a higher degree of constraint violation, and if the degrees of constraint violation are equal for two solutions, the one with the lower objective function value is considered to be the superior one.

This method of addressing constraints will ensure that any solver that compares solutions will always favour more feasible solutions and drive solutions towards greater feasibility. Once feasible solutions have been found, the solver will drive solutions towards optimality.

The method was implemented and tested on ten benchmark problems. Each solver, each individual solver tier and both benchmark methods were evaluated over ten trials and the results were compared using Wilcoxon's rank-sum test, a non-parametric statistical method that is used to test whether the medians of two distributions are equal. The time required for each solver to complete an iteration as a function of the problem size was also estimated using linear regression.

From the result of the analyses, the following conclusions were drawn:

- The two-tiered solver architecture enables a two-tiered solver that can outperform its component tiers.
- Relative performance of one local search method compared to another determines the relative performance of two-tiered solvers that use these local search methods and use the same population-based tier.
- Symmetry of the cost function of problem instances can significantly influence the effectiveness of some local search heuristics.
- Two-tiered solvers can significantly outperform the benchmark methods used in this study.
- The time required to complete iterations of the majority of the solvers that were evaluated, increases as a polynomially bounded function of the problem size.

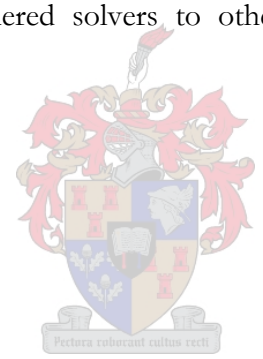
The following recommendations are made for the use and improvement of two-tiered solvers:

- Two-tiered solvers are suitable for use in practice for solving the sequential ordering problem.
- Code optimisation and alternative programming languages can be employed to improve the implementation of two-tiered solvers.
- Parameter optimisation, like varying the population size of a population-based tier, or mutation and crossover rates of genetic algorithms, can improve the performance of two-tiered solvers.

- More advanced genetic algorithms can be used to improve solver performance.

The following opportunities for further research have been identified:

- Further investigation of run-times required to identify global optima can be undertaken.
- A better understanding of the time required to perform a single solver iteration as a function of problem size, could be developed.
- The impact of the number and structure of constraints on the time required to perform a single solver iteration could be investigated.
- The distribution of candidate solutions in two-tiered solvers and how the distribution changes over iterations, could be investigated.
- The application of population-based tiered algorithms in multi-objective optimisation could be developed.
- The exploratory and exploitative behaviour found in the two-tiered algorithms could be integrated without compromising performance.
- The application of two-tiered solvers to other combinatorial optimisation problems should be developed.



OPSOMMING

Hierdie projek is aangepak om ‘n nuwe en innoverende benadering tot komplekse rangskikkingsprobleme te ontwikkel. Die projek fokus spesifiek op die ontwikkeling van ‘n metode om oplossings te vind vir die sekwensiële rangskikkingsprobleem met voorgangsbeperkings.

Die sekwensiële rangskikkingsprobleem vereis die rangskikking van ‘n aantal elemente in ‘n volgorde wat ‘n stel voorgangsbeperkings nakom en die laagste moontlike koste tot gevolg het. ‘n Voorgangsbeperking vereis dat een spesifieke element voor ‘n ander in ‘n rangskikking voorkom, en die koste van ‘n oplossing word bepaal deur die som van die kostes wat oploop as daar van een element in die rangskikking na die volgende een beweeg word.

Instansies van die probleem kom gereeld voor in die praktyk van bedryfsingenieurswese, soos byvoorbeeld: in die roete beplanning van ‘n afleweringsvoertuig, in die skedulering van take op ‘n masjien of in projek beplanning met beperkte hulpbronne.

Die sekwensiële rangskikkingsprobleem is bekend as ‘n komplekse probleem in die sin dat, met huidige beskikbare metodes, die tyd wat vereis word om ‘n optimale oplossing vir die probleem te vind exponensiëel groei as ‘n funksie van die probleem se grootte.

Die doelstelling van die projek is om ‘n oplossing te ontwikkel wat optimale of naby-optimale oplossings vind vir gevalle van die sekwensiële rangskikkingsprobleem binne ‘n prakties aanvaarbare tydperk.

Die nuwe metode funksioneer op twee interafhanklike vlakke. Die eerste, ‘n sogenaamde populasie-gebaseerde vlak, vervul ‘n ontdekkingsrol. Dit stoor kennis oor die probleem se oplossingsveld deur middel van ‘n populasie van diverse potensiële oplossings. Hierdie vlak poog om nuwe en beter areas in die oplossingsveld te identifiseer deur verandering in die populasie teweeg te bring. In hierdie projek is genetiese algoritmes en deeltjie swerm optimering gebruik as populasie-gebaseerde metodes vir hierdie vlak. Die tweede, ‘n sogenaamde plaaslike soek-vlak vervul ‘n ontginningsrol deur herhaaldelik ‘n bestaande oplossing te verbeter deur klein veranderinge aan te bring totdat geen verdere verbeterings moontlik is nie.

Die twee-vlak metode het baie in gemeen met sogenaamde memetiese algoritmes. Memetiese algoritmes is ‘n metaheuristiek wat ook populasie-gebaseerde metodes met plaaslike soek metodes kombineer.

Twee stelle doelwitte is voorgestel vir die ontwikkeling van die twee-vlak metode: Die eerste stel is prestasie-verwante doelwitte. Hierdie doelwitte vereis dat die metode optimale of naby-optimale oplossings binne ‘n prakties haalbare tyd kan identifiseer. Die prestasie van die metode ten opsigte van hierdie doelwitte word bepaal deur óf die tyd te meet wat nodig is om ‘n optimale oplossing te identifiseer, óf die koste van die beste oplossing wat binne ‘n beperkte tyd gevind kon word, te meet. Die resultate van verskillende metodes kan dan met mekaar en met ander maatstaf metodes wat in die praktyd gebruik word om die sekwensiële rangskikkings probleem aan te spreek, vergelyk word. In die projek is ‘n genetiese algoritme wat ‘n sogenaamde “lyn saamstelling” kruisingsprosedure gebruik en ‘n gesimuleerde uitgloei-algoritme gebruik as maatstaf metodes.

Die navorsingsdoelwit is om die ontginningsvermoë en ontdekkingsvermoë duidelik te onderskei en vas te stel of ‘n twee-vlak metode beter resultate lewer as wat die metodes van die twee vlakke elk op hulle eie kan doen. Dieselfde maatstawwe wat gebruik is om die prestasie van die oplossingsmetodes te meet, word gebruik om die twee-vlak oplossingsmetode se prestasie met dié van sy onderskeie vlakke te vergelyk.

Met dié doelwitte in gedagte is die volgende operasionele beginsels gespesifiseer vir die twee-vlak metode:

- lokale optima moet geïdentifiseer en gepopuleer word,
- populasie diversiteit moet aangemoedig word,
- die gemiddelde prestasie van die populasie moet verbeter word, en
- funksionele ooreenkomste tussen vlakke moet beperk word.

Om hierdie beginsels te ondersteun, is verskillende rolle aan die twee vlakke toegeken. Die volgende rolle is aan die populasie-gebaseerde vlak toegeken:

- om kennis van een iterasie na die volgende in stand te hou,
- om die ontdekkingsrol van die twee-vlak metode te vervul, en
- om probleem-onafhanklike tegnieke te gebruik, met ander woorde, om nie voordeel te trek uit spesifieke eienskappe van ‘n probleem nie.

Die volgende rolle is aan die plaaslike soek-vlak toegeken:

- om nuut gegenereerde oplossings vir identifikasie van lokale optima te optimeer,
- om voorspelbare en stabiele invloedsareas vir lokale optima te definieer, en
- om probleem-spesifieke inligting uit te buit ten einde die metode se prestasie te verbeter.

In alle implementerings van oplossingsmetodes vir die sekwensiële rangskikkingsprobleem is die volgende metodiek gebruik om voorgangsbeperkings aan te spreek:

Vir elke beperking wat oorskry word, word die graad waarmee die beperking oorskry word gekwantifiseer. Die graad van oorskryding vir 'n potensiële oplossing word gedefinieer as die som van die graad van oorskryding van elke beperking wat oorskry word. Oplossings met laer grade van oorskryding word altyd beter geag as oplossings met hoër grade van oorskryding, en as die grade van oorskryding eenders is vir twee oplossings, word die een met die laagste koste beter geag.

Hierdie metodiek verseker dat 'n oplossingsmetode wat oplossings met mekaar vergelyk altyd eers aanvaarbare oplossings sal probeer soek voordat dit oplossings se kostes probeer verlaag.

Oplossingsmetodes is geïmplementeer en getoets op tien maatstafprobleme. Tien herhalings is vir elke oplossingsmetode geëvalueer, sowel as vir elke individuele vlak en vir beide maatstaf-metodes. Die resultate is vergelyk deur gebruik te maak van Wilcoxon se rang-som toets, 'n nie-parametriese statistiese metode wat gebruik word om te toets of die mediaan van twee verdelings eenders is. Die tyd wat dit neem om 'n enkele iterasie uit te voer, uitgedruk as 'n funksie van die probleemgrootte, is dan deur linieêre regressie bepaal.

Die resultate van die analise het tot die volgende gevolgtrekkings gelei:

- 'n Twee-vlak metode is daartoe in staat om beter te presteer as die metodes van sy individuele vlakke.
- Die relatiewe prestasie van een lokale soek-vlak teenoor 'n ander dra oor na die twee-vlak metodes wat gevorm word as die vlakke gekombineer word met dieselfde populasie-gebaseerde vlak.
- Simmetrie in die koste funksie van 'n probleem kan die effektiwiteit van oplossings-metodes noemenswaardig beïnvloed.
- Twee-vlak metodes kan beter presteer as ander maatstaf-metodes wat in die studie gebruik word.
- Die tyd wat nodig is om 'n iterasie uit te voer, kan deur 'n polinoom begrens word, soos die meerderheid van die gevalle wat gedurende die studie getoets is, aantoon.

Die volgende voorstelle word gemaak vir die gebruik van en verbetering van twee-vlak oplossingsmetodes:

- Twee-vlak metodes is geskik om in die praktyk gebruik te word om die sekwensiële rangskikkingsprobleem op te los.

- Kode-optimering en implementering in ander programmeringstale kan die prestasie van twee-vlak oplossingsmetodes noemenswaardig verbeter.
- Die prestasie van twee-vlak oplossingsmetodes kan verbeter word deur die optimering van parameters wat deur die metodes gebruik word. Byvoorbeeld, die aantal oplossings in die populasie-vlak se populasie, of die kruisings- en mutasie-waarskynlikhede van genetiese algoritmes kan verander word om prestasie te verbeter.
- Meer gevorderde genetiese algoritmes kan gebruik word as die populasie-gebaseerde vlak in 'n twee-vlak metode wat beoog om prestasie te verbeter.

Die volgende geleenthede vir verdere navorsing is geïdentifiseer:

- Verdere ondersoek na die tyd wat benodig word om globale optima te identifiseer, kan onderneem te word.
- 'n Beter begrip van die tye wat nodig is om 'n enkele iterasie van 'n oplossingsmetode uit te voer as 'n funksie van die probleemgrootte, kan ontwikkel word.
- Die invloed van die aantal en struktuur van voorgangsbeperkings op die tyd wat nodig is on 'n enkele iterasie van 'n oplossingsmetode uit te voer, kan ondersoek word.
- Die verspreiding van oplossings in die populasie-gebaseerde vlak van 'n twee-vlak oplossingsmetode oor iterasies, kan ondersoek word.
- 'n Twee-vlak metode om multi-doelwit probleme aan te spreek, kan ontwikkel word.
- Die twee vlakke van 'n twee-vlak metode kan geïntegreer word sonder om die prestasie van die metode prys te gee.
- Twee-vlak oplossingsmetodes vir ander kombinatoriese optimeringsprobleme kan ontwikkel word.

TERMS OF REFERENCE

This assignment was proposed by the student as a project to fulfil a requirement for the qualification of M.Sc.Eng (Industrial).

The project was presented for approval at a departmental colloquium of the Department of Industrial Engineering at the University of Stellenbosch on 26 May 2006.

Sequencing problems are used to model scenarios in production planning, resource allocation and task scheduling; and instances of sequencing problems are regularly encountered in the practice of industrial engineering. In this thesis a specific kind of sequencing problem called the sequential ordering problem is studied.

The assignment is to develop a method for solving or approximating solutions to sequential ordering problems with precedence constraints. This method has to separate exploratory behaviour and exploitative behaviour into two separate tiers. The method has to be evaluated to determine whether or not the combination of the exploratory and exploitative tiers yields better results than the individual components. Finally, the new method has to be benchmarked against competing methods for solving the sequential ordering problem, to determine if this method could be used in practice.

The project has the following deliverables:

- Develop a new method to solve or approximate solutions to some of the sequential ordering problems. This method needs to separate exploitative behaviour that makes incremental improvements through "greedy" decisions, from exploratory behaviour that seeks out promising new areas in the solution space.
- Quantify the benefits and disadvantages that result from combining exploratory and exploitative behaviour by evaluating the different behaviours of the newly developed method.

ACKNOWLEDGEMENTS

I thank the following people and organisations for supporting this project: My study leader James Bekker for his guidance, support and patience, my mother Christene Anthonissen for proofreading this text, my wife Adri Anthonissen, who sacrificed many weekends and evenings to make this project possible and the NRF for their financial support.

Furthermore, I am grateful to many individuals whose interest and enthusiasm towards my research also kept me excited about this project, they include: The lecturing staff at the University of Stellenbosch for their interest and support of this project, my family who always had faith in me and my colleagues, both at ETP and at Hulett Aluminium where I worked while completing this project.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

basin of attraction – A basin of attraction for a local optimum is the set of solutions that will lead to this local optimum when processed by a specific local search heuristic.

Boolean variable – A variable that can take on one of two values, true or false.

edge – See "graph" (below).

fitness – A measure of likelihood of survival, the greater the fitness of an organism, the greater the probability that the organism will survive.

graph – A graph is a collection of objects called nodes and a collection of edges that each associate two nodes with one another. There are many different variations of graphs and they are often used to model real-world problems. Graph theory is a discipline in mathematics that is dedicated to exploring the properties of graphs.

Hamiltonian cycle – A term that occurs in graph theory. A Hamiltonian cycle is a set of edges that form a cycle that visits every node in the graph exactly once before returning to the starting node.

Hamming distance – A measure that is used to quantify the distance between two binary strings of equal length. The distance is measured by counting the number of locations on the strings where the binary elements differ.

Heuristic – Ideally problem-solving algorithms should have provably good runtimes and return provably good or optimal solutions. Heuristics are algorithms that give up one or both of these requirements and usually return good solutions within a good runtime, but their performance cannot be guaranteed.

meta-heuristic - This term refers to heuristic strategies that can be widely applied to solve problems although, in general, performance cannot be guaranteed.

node – see "graph" (above).

precedence constraint – This is a type of constraint found in the sequential ordering problem. This constraint requires that one element occur before another in valid solutions to a problem instance.

schema – A schema is a set of binary strings that match a certain pattern. The definition of a schema fixes certain bits to a binary value, while allowing any binary value for the remaining bits.

tabu list – This construct is used in the tabu search, a meta-heuristic, to control local search algorithms. A tabu list contains a number of operations that are forbidden or penalised if they are used. Items on the tabu list expire or become less influential as the search progresses. Operations are reinstated on the tabu list when they are used.

tractable – Problems for which solutions can be found within a reasonable period of time are called tractable problems. The usual threshold for tractability is determined by whether the time required to solve a problem instance can be bounded by a polynomial function of the problem's size. If this is not the case, the problem is said to be intractable.

transition cost – This is a cost related to moving from one element in a sequence to the next. It can also refer to the weight of an edge in a so-called weighted graph. These costs are often used to model the costs incurred by a sequence in sequencing problems, as in the sequential ordering problem or the travelling salesman problem.

1 INTRODUCTION

This project aims to develop a new and novel approach to solve complex combinatorial optimisation problems. This chapter will provide background to the problem addressed in the project; it will also set out the specific objectives of this project and the structure of the remainder of this document.

1.1 PROJECT BACKGROUND

This project addresses complex combinatorial optimisation problems that are often encountered in the practice of industrial engineering. These problems require a number of tasks or elements to be sequenced in a manner that a) ensures that the resulting sequence is feasible and b) that minimum cost is incurred in executing the sequence. Examples of instances of this problem include job sequencing in machine centres, the sequencing of destinations for a delivery vehicle or the planning of tasks in a project with limited resources.

For this particular project, the sequential ordering problem was chosen as a representative for the set of complex optimisation problems. An instance of the sequential ordering problem requires a number of elements to be arranged in sequence. The problem specifies the first and last elements in the sequence as well as a set of so-called precedence constraints, which require one specific element to appear before another in the sequence. Furthermore, the problem attaches a transition cost to each adjacent pair of elements in the sequence and determines the total cost of a solution by taking the sum of the transition costs. In solving a sequential ordering problem, the objective is to find a feasible sequence with the lowest possible transition cost.

The sequential ordering problem, like many other combinatorial optimisation problems found in practice, is intractable. This means that currently, although solutions to some problem instances can be evaluated within reasonable time periods, the time required to optimally solve these instances is not practical, regardless of the available computing resources.

When presented with an intractable optimisation problem, a decision maker will usually search for an acceptable, but not necessarily optimal, solution that can be found within an acceptable period of time. These solutions are typically found using heuristic methods that can quickly deliver results, but without a guaranteed level of performance.

In this project the aim is to develop a new heuristic solver that identifies two behavioural elements typically found in heuristic solvers and separates them into two tiers. These behaviours are a) exploitative behaviour that seeks to optimise an existing solution by making small

incremental improvements to the performance of this solution and b) exploratory behaviour that seeks to identify new promising regions of the search space.

1.2 PROJECT OBJECTIVES

The primary objective of this project is to develop a new two-tiered approach to solving complex combinatorial optimisation problems and in particular, the sequential ordering problem.

The new solver must separate exploitative and exploratory behaviour into two distinct tiers that interact in a well-defined manner to generate high-quality solutions to the sequential ordering problem within a limited time period.

The exploratory tier is called the population-based tier and will fulfil the exploratory function by maintaining a set of representative candidate solutions and by seeking to identify new promising candidate solutions that are significantly different from those in the current population.

The exploitative tier is called the local search tier and will fulfil the exploitative function by applying "greedy" search algorithms that seek to improve existing solutions through small incremental changes until no further improvements can be found.

Two significant challenges to be overcome in the development of these solvers, have been identified. These are:

- The rapidly increasing number of candidate solutions that become available as the size of problem instances increase, makes it impossible to evaluate all the possible alternatives. Successful solvers have to employ some mechanism to limit the evaluation of potential solutions without substantially compromising their ability to find optimal or near optimal candidate solutions.
- If there are constraints that limit the feasibility of solutions, a solver requires a mechanism that guides it from infeasible to feasible solutions.

The success of the new solvers will be estimated by measuring their performance on benchmark instances of the sequential ordering problem and comparing these performances to those of their individual component tiers as well as to other benchmark methods that are used to measure performance in practice.

The two-tiered solvers will be successful if they are able to outperform their individual component tiers as well as the benchmark methods currently used in practice to solve the sequential ordering problem. This project will achieve its objective by determining the success or failure of the newly developed solvers.

1.3 DOCUMENT STRUCTURE

The document is structured into six chapters. This section gives an overview of the structure and describes the content of each chapter.

This chapter gives an introduction to the project. It is intended to familiarise the reader with the general background of the project, the specific objectives of the project and the structure of the document.

Chapter 2 gives a survey of current literature on the problem definition, complexity theory and methods that are employed to address the problem. By the end of this chapter, the reader should be familiar with the literature that will be referred to in the remainder of the project.

The sequential ordering problem is defined and identified as a sequencing problem. An example of this problem is given and characteristics of the problem are explored.

Complexity theory explains how a problem's complexity can be measured. Complexity is measured in terms of the computational resources required to solve the problem as the size of problem instances grow. This theory is particularly relevant when a decision maker is faced with a time constraint and needs to choose between searching for an optimal solution, and searching for a best near optimal solution that can be achieved in a limited time. The complexity class of the sequential ordering problem is identified.

The next sections in Chapter 2 deal with existing methods that are employed to solve the sequential ordering problem and other sequencing problems.

The first two methods operate by maintaining a population of candidate solutions, where the particular set of solutions is continually adjusted to improve the overall solution quality of the population.

The first of these two methods is referred to as the genetic algorithm. Genetic algorithms simulate the natural phenomena of evolution by simulating the processes of sexual reproduction, mutation and natural selection. In this section, the history and development of the genetic algorithm is reviewed and current methods and practices used in the implementation of genetic algorithms are identified. Genetic algorithms are one of the oldest population-based methods that are still in use; therefore a significant part of the chapter is dedicated to this method.

The second population-based method is more recently developed, and is referred to as particle swarm optimisation. This method seeks to improve a population of candidate solutions by simulating the behaviour of a flock of birds. Using this method, solutions are adjusted to simultaneously approach the best previous solution found by each candidate as well as the overall

best solution found so far. The basic mechanics and some applications of this method are explored.

The last set of methods investigated in Chapter 2 is referred to as local search heuristics. These methods seek to improve existing solutions to a sequencing problem by repeatedly making small adjustments to the solution until the process is stopped or no further improvements are possible. The key components of a local search heuristic are identified and several variations to these components are reviewed.

Chapter 3 documents the design of the two-tiered solver architecture. First the objectives of the two-tiered solvers are identified. The solvers need to be competitive in terms of their performance when solving particular instances of the sequential ordering problem, and they need to be effective in separating exploratory from exploitative behaviour into two separate solver tiers. By the end of this chapter readers should understand the design and basic principles that drive the behaviour of the two-tiered solvers developed in the course of this research project.

Second, the operating principles of the solver are specified. These principles state how the solver will attempt to identify optimal or near optimal solutions to a sequential ordering problem.

Third, the structural requirements for a two-tiered solver are specified to support the operating principles. In this section, the specific roles of the two tiers and the ways in which they may interact, are identified.

The final section in Chapter 3 reviews the methods that will be used to evaluate the performances of the different solvers. This section includes the requirements and identification of suitable benchmark problems, the measures that will be used to quantify solver performance. Alternative methods to solve the sequential ordering problem against which the newly developed solvers can be compared, will be introduced.

Chapter 4 documents the implementation of specific instances of the two-tiered solvers and competing methods to solve the sequential ordering problem. By the end of this chapter, the reader should understand the specific implementation of all the solvers that are evaluated during this project.

The first section deals with specific methods and conventions used in the implementation of the two-tiered solvers. This includes the representation of instances of the sequential ordering problem, how the solvers developed in this project address constraints, and how the interaction between the two tiers of the two-tiered solvers will take place.

Three population-based tiers and four local search heuristics are implemented. In this chapter the implementation of these methods are described in detail. This includes pseudo-code that describes the program flow of each of these implementations.

The different tiers are combined to form a total of ten two-tiered solvers of which the performance will be evaluated.

This chapter also specifies the implementation of two competing methods that are used in practice for solving problems such as the sequential ordering problem.

Chapter 5 describes the evaluation process and documents the results of the evaluation of the two-tiered solvers and the alternative methods. This chapter specifies the control parameters for the evaluation and it documents the results in terms of

- the time required to execute a single iteration of each solver,
- the time required to locate a global optimum (where possible), and
- the quality of solutions found during a limited time trial.

Chapter 6 documents the conclusions and recommendations of this project based on the results presented in Chapter 5.

This chapter contains conclusions on the performance of two-tiered and other solvers developed in this project. Recommendations are made for the use and improvement of the two-tiered solvers that were developed during this project, and further research opportunities that were identified during the course of this project are documented.

The final section in this chapter summarises the work that was done in the course of the project.

Detailed results from the evaluation of the solvers as well as the code for the implementation of the solvers are attached to this document in soft copy format.

2 LITERATURE SURVEY

2.1 SCOPE

This chapter gives an overview of the literature that sets out the methods employed in this research project. The objective of the project is to evaluate the effectiveness of hybrid population-based solvers. These methods will be used in this project to develop and evaluate a new class of population-based solvers. The subjects relevant to investigating the problem are introduced as follows:

The first section of the chapter explicates the sequential ordering problem, which is the main focus of this project. The problem is defined and its properties are explored.

The second section gives a basic introduction to complexity theory. This theory is used to evaluate analytical problems in order to determine the computational resources required to solve them. With such information on computational resources a decision maker is better equipped than otherwise to decide whether he or she should seek exactly an optimal solution, or whether it would be more beneficial to approximate an optimal solution of a particular problem. At the end of the section, the complexity of the sequential ordering problem is evaluated.

The following two sections refer more directly to the project's objective by giving an overview of two population-based heuristics that can be used to address combinatorial optimisation problems, such as the sequential ordering problem (SOP). Population-based heuristics maintain knowledge about the solution space of a problem through a population of candidate solutions. Each method employs different mechanisms to generate new and potentially better solutions based on the characteristics of the current population under investigation. The two population-based methods introduced in this chapter are the best-known true population-based heuristics that maintains a fixed population of candidate solutions throughout their execution.

The third section thus reviews the theory behind genetic algorithms and in particular how such algorithms are applied to optimisation problems. The fourth section introduces particle swarm optimisation (PSO), a relatively new population-based method, which simulates the swarming behaviour of birds and insects to find solutions to optimisation problems.

In Section 2.6 on page 42, neighbourhood search methods are reviewed. In this project, these local optimisation methods are used to augment the performance of population-based heuristics when solving the sequential ordering problem.

2.2 THE SEQUENTIAL ORDERING PROBLEM

This section introduces the sequential ordering problem with precedence constraints. This problem is commonly found in optimisation problems in the industrial engineering domain and the focus of this project is the development of heuristics that attack this problem.

2.2.1 Sequencing problems

The sequential ordering problem is a kind of sequencing problem. Sequencing problems typically occur when a fixed number of elements can be ordered into a sequence and the sequence is important in that:

a) some sequences are feasible solutions to the problem and some are not, and
b) each feasible sequence has an associated cost which determines the desirability of that particular sequence.

Without any knowledge of the constraints and cost structure of a problem, each sequence would have to be evaluated individually if an optimal sequence is to be found. Since the number of possible sequences is equal to the factorial of the number of elements, the number of possible sequences can rapidly grow beyond a number that can practically be evaluated. For example:

- 5 elements allow 5! unique sequences. 5! = 120 sequences to evaluate;
- 25 elements allow 25! unique sequences. 25! = 15.5×10^{24}.

An evaluator capable of evaluating one million sequences a second will require approximately 500 million years to evaluate, one by one, each of the possible sequences from just 25 elements.

In real-world applications, the number of elements in a sequence often runs into hundreds.

Clearly, for problems larger than a very limited size, it is not practical to evaluate each of the possible sequences.

In most cases, however, some information on the constraints and cost structure of the problem is available. This enables the application of some analytical methods that give the possibility of finding good solutions without exhaustive enumeration.

2.2.2 Statement of the sequential ordering problem (SOP)

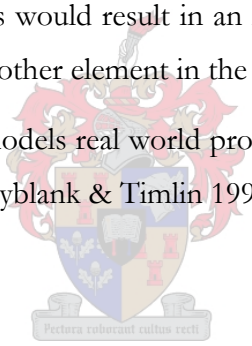
The sequential ordering problem with precedence constraints was first stated by Escudero (1988).

The sequential ordering problem relates to a number of tasks or destinations as the elements in a sequence, where the objective is to find a feasible sequence of elements that incur the least cost.

In the sequential ordering problem the cost of a sequence in the particular problem is defined as the sum of the transition costs incurred when proceeding from one element in the sequence to the next. The cost structure of a sequential ordering problem can be described by an $n \times n$ matrix that contains the transition costs between each pair of elements. This is considerably more compact than calculating the possible $n!$ unique costs required if each sequence is assigned an independent cost of its own.

A sequential ordering problem can also be subject to precedence constraints. A precedence constraint requires that a particular element precede another in a sequence. To ensure that a feasible solution exists, the precedence constraints cannot form a cycle, *i.e.* *A* precedes *B*, *B* precedes *C* and *C* precedes *A*. This would result in an impossible situation where every element in the cycle needs to precede every other element in the cycle.

The sequential ordering problem models real world problems like machine scheduling (Escudero 1988) or single vehicle routing (Pullyblank & Timlin 1991, Savelsberg 1990).



An example of the sequential ordering problem applied to a single vehicle routing problem:

This example models a so-called drop-off and delivery problem. A delivery truck starts out at a depot and needs to pick up and deliver a number of parcels at a number of locations before returning to the depot.

Some parcels arrived at the depot earlier and so are picked up prior to starting the trip, but others have to be picked up along the route. A parcel cannot be delivered before it has been picked up.

A schematic of the delivery destinations can be seen in Figure 1. Roads are indicated as lines, squares indicate locations at which the vehicle makes its stops. Locations A and H represent the start and end locations respectively.

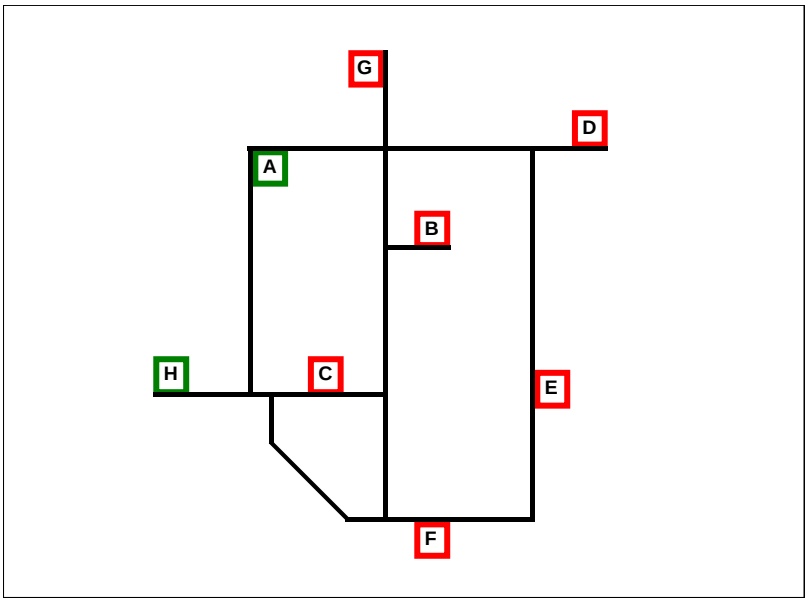


Figure 1: Schematic View of Delivery Destinations for a Single Vehicle Routing Problem

In this particular problem, the vehicle needs to pick up a parcel at location E for delivery at location B and another at location D for delivery at location C. All other destinations are delivery locations for parcels that have previously been received at A.

The constraints are shown in Table 1.

Constraint Number	Preceding Element	Succeeding Element
1	E	B
2	D	C

Table 1: Precedence Constraints for the Sequential Ordering Problem Example

The distance between locations determines the transition costs, as shown in Table 2.

		Destination							
		A	B	C	D	E	F	G	H
Origin	A	-	5	8	9	13	12	4	8
	B	5	-	6	9	13	10	5	11
	C	8	6	-	14	5	4	10	4
	D	9	9	14	-	8	13	8	17
	E	13	13	5	8	-	5	13	12
	F	12	10	4	13	5	-	14	7
	G	4	5	10	8	13	14	-	12
	H	8	11	4	17	12	7	12	-

Table 2: Transition Cost Matrix for the Sequential Ordering Problem Example

A valid solution to this problem of finding the optimal pick up and delivery route, can be represented as a sequence that starts with A, ending with H, where E occurs before B, and D occurs before C. The cost associated with each solution is determined by summing the transition cost between every adjacent pair of elements in the sequence. For instance, the cost of a potential solution {A,G,D,E,F,B,C,H}is determined as is shown in Table 3:

Origin Element	Destination Element	Transition Cost
A	G	4
G	D	8
D	E	8
E	F	5
F	B	10
B	C	6
C	H	4
Total Cost		**45**

Table 3: Example of Cost Calculation for a Solution to a Sequential Ordering Problem

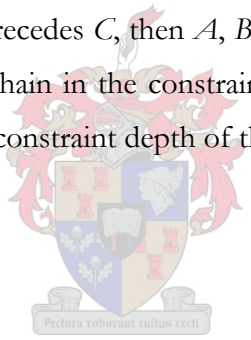
Using the information contained in the transition cost matrix and the constraint list, potential solutions can be validated to determine whether or not they satisfy the precedence constraints and can be evaluated to determine their associated costs.

This information can be given as input to algorithms, which will search for sequences that satisfy the given constraints and have the lowest overall cost.

2.2.3 Special properties of sequential ordering problems

The general case of sequential ordering problems (SOPs) provides knowledge about these kinds of problems that allows for more efficient search methods than blind enumeration of sequences. Many real world sequential ordering problems have additional properties that can be exploited by appropriately designed search methods. Such properties include:

- **Symmetry** – If the transition cost from element *A* to element *B* is the same as the transition cost from element *B* to element *A*, for any two elements *A* and *B*, the sequential ordering problem is said to be symmetric.
- **Metric** – If the transition cost from element *A* to element *B* is less than or equal to the sum of the transition costs from element *A* to element *C* and element *C* to element *B*, for any combination of elements *A*, *B* and *C*, the cost function and the sequential ordering problem is said to be metric.
- **Maximum Constraint Depth** – The longest chain of precedence constraints in a sequential ordering problem defines the constraint depth of the problem, *i.e.* if *A* precedes *B* and *B* precedes *C*, then *A*, *B* and *C* form a chain of three elements. If this is the longest chain in the constraint set of an sequential ordering problem, then the maximum constraint depth of the sequential ordering problem is three.



2.3 COMPUTATIONAL COMPLEXITY

One of the decisions that face an investigator seeking to solve an optimisation problem is whether to search until an optimal solution is found, or to approximate the optimal solution using a heuristic device. This section reviews the tools made available by complexity theory to evaluate the resources required to solve optimisation problems and make such a decision.

The study of computational complexity deals with the computational resources, in particular processing time and memory, that is required to execute algorithms.

This section provides an overview of concepts found in the first chapter of Ausiello *et al.* (1999).

2.3.1 Measuring complexity

In order to quantify the complexity of an algorithm, a measure is required that will assist in specifying the time it takes to execute the algorithm and the memory or information storage capacity required during the execution of the algorithm.

The time required for executing an algorithm is determined by various factors such as the hardware on which the algorithm is executed, the efficiency of the code, and the inputs to the algorithm.

This means that a measure independent of factors external to the algorithm itself, has to be used.

Asymptotic analysis

Asymptotic analysis provides a useful means of measuring the performance of an algorithm as the size of the input tends to infinity. The size of the input can be measured as the number of binary bits that is required to represent it.

If the amount of a computational resource required, is expressed in terms of the size of the input, n, by a function f, then asymptotic analysis of f enables us to understand how $f(n)$ behaves as $n \to \infty$.

The following notation is used to express the asymptotic behaviour of a function:

It is said that $f(n)$ is $O(g(n))$ if there exists constants a,b and n_0, such that, for all $n \geq n_0$,

$$f(n) \leq cg(n) + a \qquad \ldots(2.1)$$

$O(g(n))$ describes the worst case performance of $f(n)$ as n becomes large.

Similarly, it is said that $f(n)$ is $\Omega(g(n))$ if there exists constants a,b and n_0, such that, for all $n \geq n_0$,

$$f(n) \geq cg(n) + a \qquad \ldots(2.2)$$

Ω ($g(n)$) describes the best case performance of $f(n)$ as n becomes large.

If $f(n)$ is $O(g(n))$ and $f(n)$ is $\Omega(g(n))$, then it is said that $f(n)$ is $\Theta(g(n))$.

Although the actual behaviour of f could be very difficult to describe and can vary according to the actual input (not just the length of the input), the asymptotic behaviour allows one to put bounds $f(n)$ as n grows larger.

When analysing the complexity of an algorithm, the worst possible complexity of the function is used most often for the following reasons:

- worst case performance of the algorithm guarantees a minimum level of performance, and
- when analysing the complexity of all algorithms that solve a particular problem, O complexity is often easier to determine than Ω complexity (Ausiello *et al.* 1999, p. 9).

2.3.2 The classification of problems

Once the means are available to measure the complexity of an algorithm, it becomes possible to measure the complexity of problems.

Problem P is defined as the combination of a set of problem instances I_P and a set of problem solutions S_P. When analysing the complexity of a problem, a member of I_P is given as the input and a member of S_P is expected as output (provided that the input is in fact a member of I_P).

a.) Complexity of decisions problems

A problem P is called a decision problem if the set I_p, of all instances of P can be partitioned into a set Y_p of all positive instances of P, and a set N_p of all negative instances of P.

The solution set of P has only two elements, s_p and s_n so that $P \subseteq (s_p \times Y_p) \cup (s_n \times Y_n)$.

An algorithm A that solves P returns a YES answer if the input $x \in Y_p$ and a NO answer if $x \in Y_n$. The behaviour of A is undefined where $x \notin I_p$.

An example of a decision problem is the satisfiability problem:

Problem Definition – SATISFIABILITY:

Instance: A formula F in the conjunctive normal form on a set of Boolean variables V.

Question: Does there exist an assignment of V so that F evaluates to TRUE.

For instance: Given Boolean variables *a*, *b*, *c* and *d*, does there exist a set of truth assignments for *a*, *b*, *c* and *d* that satisfies *F*, where $F = (a + \bar{c} + d) \cdot (a + \bar{b} + \bar{c}) \cdot (\bar{b} + c + d) \cdot (a + b + c)$.

For this example, it is practical to enumerate all 2^4 possible combinations of assignments and evaluate *F*, but as the number of Boolean variables increases, the number of possible combinations increases exponentially, and the enumeration would become impractical.

Decision problems can be classified by the complexity characteristics of the algorithms required to solve them. For example, if an algorithm *A* exists that solves *P,* and *A* has time complexity $O(g(n))$, then *P* has time complexity $O(g(n))$. Similarly if no algorithm exists with a time complexity better than $\Omega(g(n))$ that solves *P*, *P* has time complexity $\Omega(g(n))$.

The complexity of problems can be classified according to complexity classes. For example, the following kinds of classes can be identified:

P is the class of all problems that can be solved in time proportional to a polynomial of the input size, *i.e.* a problem is a member of P if and only if the time complexity of the problem is $O(g(n))$, where $g(n)$ is a polynomial.

Pspace is the class of all problems that can be solved using memory proportional to a polynomial of the input size, *i.e.* a problem is a member of Pspace if and only if the space complexity of the problem is $O(g(n))$, where $g(n)$ is a polynomial.

Exptime is the class of all problems that can be solved in time proportional to an exponential of the input size, *i.e.* a problem is a member of Exptime if and only if the time complexity of the problem is $O(g(n))$, where $g(n)$ is an exponential.

It can be shown that the satisfiability problem belongs to the Pspace class:

Any instance of the satisfiability problem can be checked by enumerating all possible solutions and checking to determine whether one satisfies the problem formula or not. The memory required to store the enumerated solutions can be used again for each enumeration and need not exceed the size of the number of Boolean variables in the problem formula, hence the memory required, is $O(n)$.

The same method also confirms that the problem is in Exptime, as the number of enumerations would be equal to 2^n, where *n* is the number of Boolean variables. At the time of writing it is not known whether or not the problem is also a member of P.

The class P is traditionally considered to be a reasonable threshold between tractable and intractable problems (Ausiello *et al.* 1999, p. 11).

b.) Complexity class NP and NP completeness

One class of problems that is particularly important in complexity theory, is the class **NP**.

NP is the class of all problems that can be solved using a non-deterministic algorithm in time proportional to a polynomial of the input size.

A non-deterministic (ND) algorithm is a theoretical concept that cannot be executed on any existing computer. The execution of an ND algorithm would require a computer with an unlimited number of parallel processing units. An ND algorithm can execute an instruction that branches the execution of the algorithm onto two different processing units and sets one variable used by the algorithm to a different value for each branch. There is no limit to the number of branching instructions an ND algorithm can execute. If any branch terminates and returns a YES value in response to a decision problem, the algorithm will return a YES solution. If no branches return a YES value, the algorithm will return a NO solution.

An ND algorithm can solve the satisfiability problem by branching for each Boolean variable in the problem formula and testing the end result of each branch. This algorithm will evaluate the satisfiability problem with time complexity $O(n)$.

The class NP contains all problems that can be solved in polynomial time using an ND algorithm.

One important open problem at the time of writing arises from the question whether or not class P is equivalent to NP. It would be an impossible task to test all instances of NP to see whether they also reside in P, but Cook (1971) provided a route for addressing this problem.

Cook proved the existence of a subset of NP called the NP-complete set:

If an algorithm exists that can transform any instance x of decision problem P_1 into an instance of decision problem P_2 in such a way that $x \in Y_{P_1}$ if and only if $x \in Y_{P_2}$, then P_1 is said to be reducible to P_2. A problem P is said to be NP-complete if any problem in NP is reducible to P in polynomial time.

Cook further proved that the satisfiability problem was such an NP-complete problem.

Once a problem is known to be NP-complete, it is possible to prove that another problem P is a member of the NP-complete set using the following procedure:

a) Prove that P is in NP, by specifying an ND algorithm to solve P.
b) Prove that any one NP-complete problem can be reduced to P in polynomial time.

If the second condition can be proved, P is said to be NP-hard.

Since the satisfiability problem has been shown to be NP-complete, many other problems have been proven to belong to this set.

Currently, after many years of research, the question of whether or not P=NP is still open. In fact, it is considered to be one of the most important open problems in mathematics today.

c.) Complexity of optimisation problems

Optimisation problems can be defined as follows:

An optimisation problem P, is characterised by a set of combinations of the following objects:

- A set of instances, I_P.
- A set of solutions, SOL_P.
- A measure function, m_P defined over pairs (x,y), where $x \in I_P$ and $y \in SOL_P$. For each pair where y is a feasible solution to x, $m_P(x,y)$ provides the value of the solution y to problem instance x.
- A goal indicator, $goal_P$, which indicates whether the objective is to maximise or minimise the measure function.

For each instance I_P, we denote SOL_P^* as the set of optimal solutions to I_P, that is the set of solutions whose value is optimal (either minimal or maximal, depending on $goal_P$) and where $m_P(x)^*$ denotes the value of the optimal solutions of x.

For any optimisation problem P, there are at least three computational challenges:

1. The **Constructive Problem** (P_C) – Given $x \in I_P$, derive an optimal solution for $y^*(x) \in SOL^*(x)$ and its measure $m^*(x)$.
2. The **Evaluation Problem** (P_E) – Given an instance of $x \in I_P$, derive the value of $m^*(x)$.
3. The **Decision Problem** (P_D) – Given an instance of $x \in I_P$, and a positive integer $K \in Z^+$, decide whether $m^*(x) \geq K$ if $goal_P = MAX$ or if $m^*(x) \leq K$ otherwise.

For optimisation problems, the most relevant question is whether the constructive problem is tractable or not, *i.e.* whether a polynomial time algorithm exists that can construct an optimal solution and its associated value.

From a complexity point of view, optimisation problems can be classified as follows:

The class **NPO** is defined as the set of problems where:

- The set of instances I_P, is recognisable in polynomial time.
- Given an instance of I_P and a potential solution y, it is decidable in polynomial time whether or not $y(x) \in \text{SOL}(x)$.
- The measure function m can be computed in polynomial time.

Furthermore, **PO** is defined as the subset of NPO for which a polynomial time algorithm exists that can solve the construction problem P_C.

It can be shown (Ausiello *et al.* 1999, p. 28-29) that for any optimisation problem P in NPO, the corresponding decision problem P_D, is a member of NP. Furthermore, it can be proven that if NP = P, then also NPO = PO and *vice versa*. Since a great many optimisation problems, for which no polynomial time algorithms have been found, are known to be in NPO, the question of whether NP = P becomes important to determine whether polynomial time algorithms that can solve optimisation problems in NPO exist at all.

Finally if the decision problem P_D is known to be NP-complete, then P can only be a member of PO if NP = P. So if it can be shown that P_D is NP-complete, the most practical way of attacking the optimisation problem would be using a heuristic or approximation algorithm.

2.3.3 The complexity of the Travelling Salesman Problem and the Sequential Ordering Problem

The travelling salesman problem or TSP, is a classic problem in operations research. The TSP is easy to state, but the most general case of the TSP is NPO-complete, meaning that any problem in NPO can be reduced to a TSP in polynomial time (Orphonen & Mannila 1987).

The general TSP can be stated as follows:

The travelling salesman problem

Objective: Minimise the length of a tour of m cities.

Instance: A set C of m cities, distances $d(c_i,c_j) \geq 0$ for each pair of cities.

Solution: A tour of C, *i.e.* a permutation π of all the cities in C.

Measure: The length of the tour: $d(c_{\pi(m)}, c_{\pi(1)}) + \sum_{i=1}^{m-1} d(c_{\pi(i)}, c_{\pi(i+1)})$

The general TSP can be characterised by the properties of the distance function:

- If $d(x,y) = d(y,x)$, the problem is called a symmetric travelling salesman problem or STSP, otherwise the problem is called an asymmetric travelling salesman problem or ATSP.
- If $d(x,z) \leq d(x,y) + d(y,z)$, the problem is called a metric travelling salesman problem.

Unlike the general case, the metric TSP is not NPO-complete.

The sequential ordering problem (SOP), as stated by Escudero (1988), can be formally described by its objective, instances, solutions and measure. It reads as follows:

The sequential ordering problem:

Objective: Minimise the cost of m operations.

Instance: A set O of m operations, costs $c(o_i,o_j) \geq 0$ for each pair of operations and a set of precedence constraints P that does not form a cycle on the operations.

Solution: A permutation π of all the operations in O that satisfies the precedence constraints in P.

Measure: The cost of π: $\sum_{i=1}^{m-1} d(c_{\pi(i)}, c_{\pi(i+1)})$

The sequential ordering problem is a general case of the TSP and is therefore also NPO-complete.

2.4 GENETIC ALGORITHMS

Genetic algorithms (GAs) are a collection of methods used to solve control and optimisation problems by simulating evolution. They have been used to address a wide variety of problems ranging from numerical and combinatorial optimisation, engineering design and machine learning to modelling economic and ecological phenomena (Mitchell 1996, pp.15-16).

The GA was one of the first population-based methods applied to solving difficult optimisation problems and the application of GAs to these problems has been research extensively. Since the GA is one of the oldest and most widely used population-based methods today, it is an important part of this project and will be reviewed in detail in this section.

GAs were first proposed by Fraser (1957a and 1957b). John Holland (1975) provided the first theoretical analysis of GAs, but they only started to receive serious attention from researchers during the last two decades. This section will give an introduction to the origins and the current development of genetic algorithms. The focus is on the role of genetic algorithms in optimisation.

2.4.1 Biological background

The idea of GAs was inspired by the way evolution works in nature. In particular, they are conceptualised as mimicking the molecular processes that are observed in natural evolution.

a.) Evolution in nature

Living organisms consist of cells, and each cell contains one or more chromosomes that describe the organism. Each set of chromosomes serves as a "blueprint" for the organism. The chromosomes contain all the information used to construct the proteins that the organism requires. Accordingly, the chromosomes can be partitioned into genes, where each gene provides a blueprint for a particular protein. Genes determine specific characteristics of individual organisms, for instance, hair or eye colour. The different possible configurations of a gene are called *alleles*. The position of a gene on a chromosome is called its *locus*. A complete set of chromosomes for a particular organism is called its *genome*. The components and concepts that describe a genome are demonstrated in Figure 2.

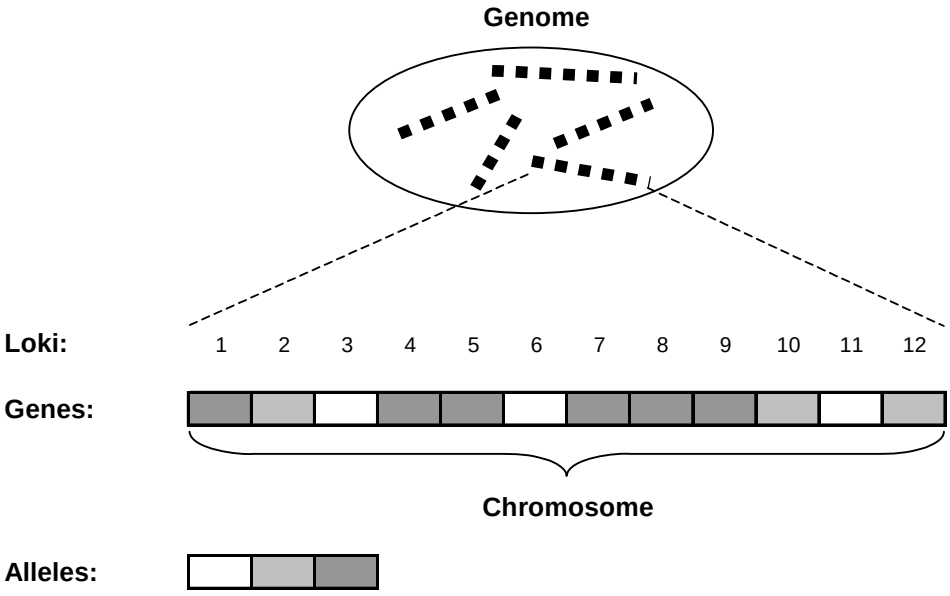


Figure 2: A Conceptual Example of a Genome

Evolution changes the characteristics of organisms through two processes:

- mutation, the introduction of random changes to the genome via replication errors, and
- recombination, the composition of a new genome by the merger of two others during sexual reproduction.

New genomes give rise to new organisms that are either more or less likely to survive and reproduce than their parent(s). The likelihood of survival (and reproduction) of an organism in a particular environment is referred to as the *fitness* of the individual organism.

The more fit an individual organism, the more likely it is to survive and propagate its genes to future generations. Furthermore, the environment in which an organism finds itself can exert greater or lesser pressure on the organism: In an environment with abundant resources and few dangers, all organisms would be likely to survive, while more hostile environments will demand a greater degree of fitness from surviving individuals.

This effect of the environment on the population of organisms that inhabit it, is called *selective pressure*. The selective pressure determines the degree of discrimination between organisms with different levels of fitness.

Thus an evolutionary process acts on a population of individual organisms to reproduce their genes, and the selective pressure of the environment drives new surviving individuals to be more likely to survive than their predecessors.

b.) Evolution in computation

The concept of genetic algorithms (GAs) introduced through Holland's original framework for an adaptive plan, is based on natural evolution by its encoding of artificial organisms into binary strings. These new organisms are evaluated according to specific criteria. Such evaluation, determines the fitness of individual organisms and on this basis a limited number of these individuals are selected, according to their fitness, to form the next generation. This process is repeated until some terminating condition is met.

Strings are processed by mimicking the natural processes of mutation and recombination.

The first process that is used in the original GAs is called *crossover*. During this process, a number of strings are paired off to "mate". For the mating procedure an index on the binary string, the crossover point, is selected and two new strings are composed by interchanging the parts of the strings on the "parents" after the index. An example is schematically given in Figure 3

The second process that Holland later introduced is called *mutation*. *Mutation* introduces a probability for each bit in each string in the population to be inverted. This process is called *mutation* since it can be considered to be analogous to mutation in nature.

.

Crossover is analogous to recombination in nature.

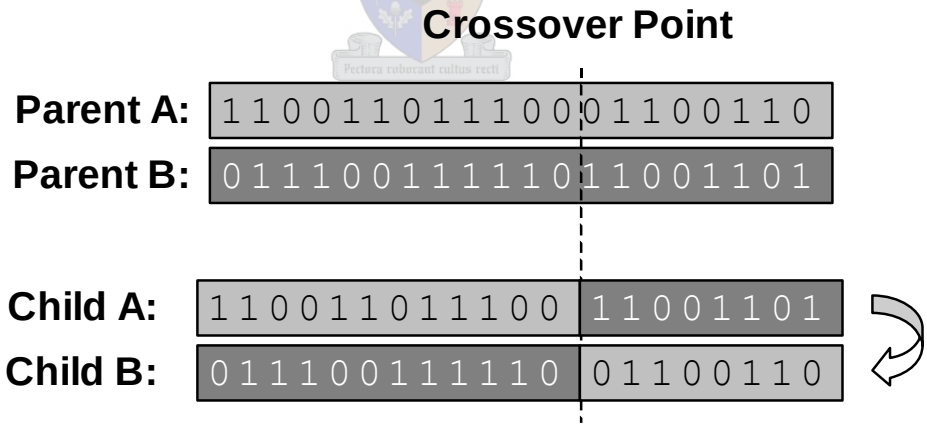


Figure 3: A Example of one-point Crossover

Borrowing the terms used in genetics, the encoding of an artificial organism is called a *genome* (or *chromosome*), each bit is called a gene and each particular setting for the bit (0 or 1) is called an *allele*. The position of a bit in the binary string is called its *locus*.

Since the original GA was introduced by Holland, many different encodings and genetic operators have been introduced, that share a similar strategy.

2.4.2 Mutation, Crossover and Representation

The two most common genetic operators are the two referred to above, namely mutation and crossover. This section will investigate operators to discover how these operators construct new solutions. The way in which genetic operators influence the performance of new individuals is strongly related to the representation of the individuals. Different possible ways to represent individual artificial organisms will be discussed.

a.) The Roles of Mutation and Crossover

Holland (1975, p. 111) argued that crossover was the primary means by which GAs construct new potential solutions and find promising patterns, whereas mutation was merely a disruptive background operator to provide insurance against lost alleles.

An empirical investigation by Wu, Lindsay and Riolo (1997) tested these assumptions and found some surprising results. They measured the construction and destruction of "building blocks" in the population by each operator and also measured the diversity of individuals in the population. *Building blocks* are patterns in the genome whose presence improves its fitness. (See Subsection 2.4.4d.) on page 33) for an overview of the building block hypothesis.

Wu *et al* (1997) found that mutation was just as good or better than crossover in constructing building blocks, but, as could be expected, mutation was also much more destructive than crossover.

However, crossover played a key role in propagating newly found building blocks throughout the population, and did so very effectively. In fact, it was observed that as soon as new basic building blocks were discovered, they were quickly propagated to almost all the individuals in the population, drastically lowering diversity in the population.

Wu *et al*'s (1997) research illustrated that, mutation is not necessarily just a background operator, but can play a significant role in the performance of a GA.

b.) Representation and Alternative Operators

The way in which individual genomes are represented in the population of a GA, is called an *encoding*. An encoding maps the representation of an individual to an actual individual. Goldberg (1989a) argued that the low number of alleles (1 and 0) in binary encoding should theoretically enhance the performance of a GA and he questioned the effectiveness of non-binary encodings in GAs.

Antonisse (1989) opened the door to new applications for GAs by proving that the "implicit parallelism" of GAs does not depend on a binary string representation. Since then others

(Wright, 1991 and Davis, 1991) empirically verified the effectiveness of non-binary encodings as opposed to binary encodings.

New encodings and alternative forms of crossover and mutation can be used to tailor GAs to a specific problem. Without looking at any specific encodings, general principles for encoding individuals as potential solutions, are investigated. An encoding may be classified using the following categories[1].

1. Isomorphic Encodings

In this type of encoding each feasible individual has exactly one possible representation and each possible representation maps to a feasible individual. So there is a one-to-one correspondence between feasible individuals and possible representations.

2. Incomplete Encodings

An incomplete encoding is unable to represent all possible feasible individuals. Such an encoding is generally undesirable, unless the fitness of representable individuals can be guaranteed in some way to be better than other feasible individuals.

3. Constrained Encodings

A constrained encoding has possible representations that do not map to any feasible individuals. These representations are called illegal representations. This can cause problems for a GA. If feasible individuals in such an encoding can only produce feasible offspring when operated on by the operators of the GA, the encoding is called a closed constrained encoding. Such encodings can still be useful, provided a method exists to generate feasible individuals to start with.

4. Redundant Encodings

Redundant encodings can map more than one representation to a single individual. It is possible that the genetic operators do not treat different representations of the same individual equally.

Except for isomorphic encodings, these encodings are not mutually exclusive.

Constrained encodings are often required when the search space of a problem is arbitrarily constrained. Such problems can be troublesome for GAs. Most methods for dealing with constraints are based on a penalty function that assigns poor fitness to infeasible individuals, thereby decreasing their occurrence in a population. Several ways of dealing with infeasibility are discussed by Michalewicz (1992).

---

[1] These categories were not found in GA literature. They are introduced here by the author to classify charateristics of encodings.

Redundant encodings are mostly not a problem (unless they are incomplete or constrained), but they may lead to inefficient computation.

For any complete encoding, genetic operators can be devised that mimic the performance of any other genetic operators on a different encoding by transforming the first encoding to the second one, applying the second set of operators and transforming the results back to the first encoding. So the combination of the encoding and its operators determine the operation of the GA.

By themselves, encodings are important in that they give insight into the structure of individual solutions and suggest means to manipulate this structure. A good encoding should facilitate the computation of the genetic operators.

2.4.3 Selective pressure and diversity

Many different encodings and operators have been developed for various applications, but two primary factors always influence the evolution process in GAs: (i) population diversity and (ii) selective pressure (Michalewicz, 1992, p. 58). These two are related: high selective pressure decreases diversity and limits the construction of new solutions, while weak selective pressure does not give enough credit to good solutions and prevents their development. Low pressure encourages exploration of the search domain, while high pressure exploits the already fit individuals to find local optima.

a.) Selection for Survival

The selective pressure of an environment guides organisms to adapt to their surroundings. In GAs the environment is partially represented by the fitness function, but the selection method exerts pressure.

Some selection methods use the value of the fitness function directly to compare and select individuals. These methods are sensitive to the shape of the fitness function. The shape of the fitness function can be visualised by plotting the fitness of all the candidate solutions in increasing order of fitness. See Figure 4 for examples of different fitness functions.

The fitness function in Figure 4 a) shows a concave fitness function. As the fitness of the individuals in the population increases, the differences in the fitness of population members are expected to decrease as the function levels off and it will become more difficult for selection methods that are sensitive the shape of the function to discriminate between more and less fit individuals in the population. Figure 4 b) shows the opposite. This convex fitness function will lead to greater discrimination between individuals as the overall fitness of the population increases.

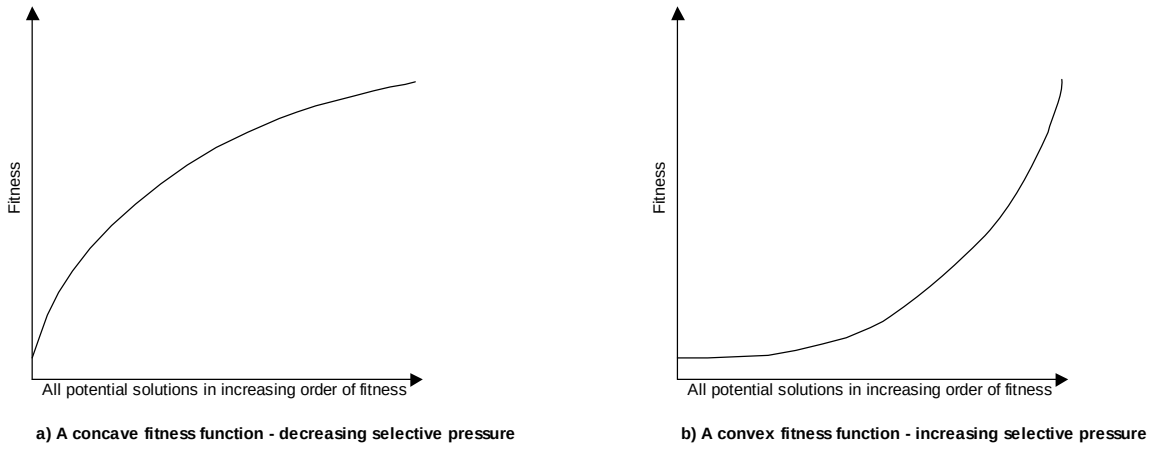


Figure 4: Shapes of fitness functions

Several selection methods have been proposed and some of these are reviewed here:

Deterministic Sampling is, computationally, the simplest method of selection. To select *n* individuals using deterministic sampling, individuals are ranked according to their fitness and the *n* fittest individuals are selected. Individuals can only be selected once using this method. Deterministic sampling is not sensitive to the shape of the fitness function and only takes the fitness rank of the individuals into account.

The Roulette Wheel method, as it is now known, is the original method used by Holland (1975) to select individuals. This method is described in detail in Subsection 2.4.4b.) on page 29. Using this method, the likelihood of an individual being selected is equal to the fitness of the individual in proportion to the total fitness of the population, and each individual may be selected multiple times. This method is similar to assigning a number of slots on a roulette wheel to individuals in proportion to their fitness and spinning for a winner. This method is sensitive to the shape of the fitness function.

Stochastic Universal Sampling (SUS) was proposed by Baker (1985) to reduce the variability inherent in roulette wheel selection. Like the roulette wheel, this method assigns probabilities to individuals based on their fitness, but rather than spinning the wheel *n* times to select *n* individuals, this method is equivalent to spinning a wheel with *n* equally spaced pointers; it simultaneously selects all the individuals for the next generation. The principle is shown in Figure 5. This method will guarantee the selection of individuals with above average fitness. One weakness of this method is that the location of individuals on the wheel will affect the likelihood of individuals being selected together, *i.e.* if two neighbours on the wheel have a total fitness less than the population mean, they cannot be selected simultaneously. Like the roulette wheel method, SUS is sensitive to the shape of the fitness function.

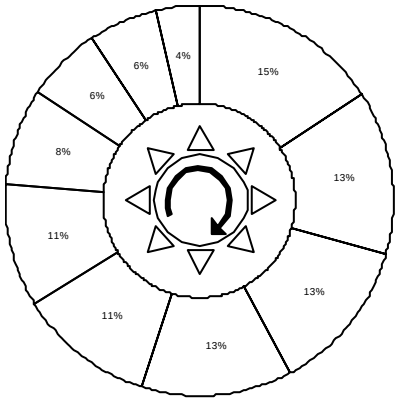


Figure 5: An Example of Stochastic Universal Selection

Sigma Scaling is described by Mitchell (1996, pp.167-168). Sigma scaling assigns each individual an expected selection value according to the difference between its fitness and the population mean and divided by a multiple of the standard deviation of the population fitness. Individuals then have a probability of selection that equals the proportion of their expected selection value to the sum of all expected selection values in the population. Using sigma scaling, the scaling constant of the standard deviation controls selective pressure. Sigma scaling ensures that individuals that have a very high fitness, do not dominate the selection process and such scaling also causes above average individuals to stand out when there is little variation in the population fitness. Sigma scaling is sensitive to the shape of the fitness function.

Boltzmann Selective Pressure is a method that is used to vary selective pressure over the generations of a genetic algorithm. This method was inspired by the simulated annealing heuristic and was introduced by de la Maza and Tidor (1991). Typically, low selective pressure is used in earlier generations to encourage exploration of the solution space; selective pressure is slowly increased as the algorithm progresses, to force convergence on good solutions.

Ranking (Baker 1985) has become an increasingly popular method to control selective pressure and to make selection independent of the shape of the fitness function. Rank selection sorts all individuals by their fitness and then assigns a probability of selection to each individual according to its rank. The specific assignment of probabilities to rank can vary with the implementation. This method obscures the absolute differences in fitness and makes selection independent of the shape of the fitness function.

Tournament Sampling (Goldberg & Deb 1991) is another method that can be used to obscure absolute differences in fitness between individuals, and it is computationally more efficient than ranking. Using tournament sampling, two or more individuals are selected from the population at

random. The method then selects one of the individuals with a higher probability of selection given to fitter individuals.

The Elitist Model was developed by de Jongh (1975) and is always used in combination with another selection method. When elitism is used, a percentage of the best performing individuals in the population is always selected for the next generation.

b.) Protecting Diversity

Convergence occurs when particular characteristics tend to spread across the entire population and individuals tend to lose their unique characteristics. One threat to the performance of a GA is premature convergence. Premature convergence occurs when the GA does not have sufficient opportunity to explore its solution space before the population converges to a single, potentially sub-optimal solution. The following methods have been suggested to prevent premature convergence:

The Crowding Factor Model was developed by de Jongh (1975). With this model, new individuals are continually generated to replace existing individuals in the population. The individual to be replaced is selected from a group of individuals that most resemble the newcomer.

Serial Selection was developed as a mechanism for protecting diversity, by Goldberg (1989b). This method repeatedly runs small populations to convergence, selects a group of the best individuals and reinitializes the remaining population with new individuals.

The Rank-Space Method, proposed by Winston (1993 pp. 505-528), uses a diversity principle. According to the diversity principle it is as good for an individual to be different as it is to be fit. During selection, each candidate is compared to the population of already selected individuals and is ranked according to how different it is to other individuals. Each individual is also ranked according to fitness. The two rankings are then combined to rank the individuals again. This final ranking is used to determine an individual's probability of selection. To enable this method, there has to be some measure of distance between individuals that can be used to measure similarity. Winston proposes that individuals in a GA using the Rank-Space method will tend to avoid convergence and attempt to find the global maximum by identifying and populating the local maxima.

2.4.4 Schemata and building blocks

This section describes the theory behind Holland's original GAs, as well as some further developments in the understanding of how GAs function.

To solve an optimisation problem, a GA represents potential solutions to a problem as individuals in a population of artificial organisms.

GAs were designed to create and identify "good" individuals in a population, encoded as binary strings of fixed length, and to combine them to form better individuals. The "goodness" of an individual is determined by an objective function that maps any particular individual to a positive number, called the fitness of the individual. For a GA to function meaningfully, good individuals have to be combined in such a way that their "goodness" can be preserved. If the fitness of the individual is determined by presence of specific binary sequences in its representation, such sequences can be represented by schemata.

a.) What is a schema?

A binary string of length n, can have 2^n possible values. However, if we decide to fix k of the values, we have a set of $2^{(n-k)}$ possible strings. Such a set is called a schema. A schema can be represented by a string of 1's, 0's and □'s, where □ represents a "don't care". Each non-□ element in the string is called a defined bit. For example: The string (□□□□□) would represent the set of all binary strings of length 5 with no defined bits, and (101□□) would represent all strings of length 5, starting with 101, the three defined bits.

The number of fixed elements in a schema is called the order of the schema, for instance (101□□) is a schema of order 3, whereas (1□□0□) is a schema of order 2. The order of a schema, S, will be denoted by $o(S)$.

The defining length of a schema is the difference between the loci of the first and the last defined bits. So, the schema (101□□) would have a defining length of 2 and (1□□0□), would have a defining length of 3. We will denote the defining length of a schema, S, by $\Delta(S)$.

Any string that is included in a schema is said to be an instance or a sample of the schema. So, (10110) would be an instance of (101□□), but not of (1□□0□), whereas (10101) would be an instance of both (101□□) and (1□□0□).

As with binary strings, fitness can also be assigned to schemas. The fitness of a schema is defined as the average fitness of all the binary strings in the schema. Although this is a sound definition for the fitness of schemata, it requires that all instances of a schema be evaluated to determine the fitness of the schema. This makes evaluating low order schemata very inefficient.

b.) How do GAs process schemata?

In this section, the process of the original GA as it was proposed by Holland is discussed. Since GAs were first proposed, many variations have been introduced that differ considerably from the process he suggested. However, Holland's basic theory on how these GAs process schemata remains a point of departure. Therefore his theory is presented here.

According to Holland (1975: 66-74) GAs do not directly process schemata, but instead they process individual strings in a manner that guarantees the propagation of good schemata. In the following subsection the process of the GA is described and the propagation of schemata as they are processed by the GA is analysed. Finally, the Schemata Theorem is presented, which is the theoretical cornerstone of Holland's adaptive plans.

The GA attempts to identify strings, also referred to as individuals, that perform well with respect to some fitness function *f*. The function *f* maps each individual to a non-negative real number that is called the fitness of the individual. The GA operates on a population, P_t of binary strings $S_1,S_2,...,S_n$ of equal length to produce P_{t+1}. The size of the population remains constant. Total population fitness F is defined as the sum of the fitnesses of all the individuals in the population and the average population fitness $\overline{F}$ as the arithmetic mean of these fitnesses:

$$F(P_t) = \sum_{i=1}^{n} f(S_i) \qquad ...(2.3)$$

$$\overline{F}(P_t) = \frac{\sum_{i=1}^{n} f(S_i)}{n} \qquad ...(2.4)$$

Here n represents the size of the population.

At the start of each generation, potential members of the new generation are selected from the old generation in the following manner:

Each string is evaluated according to the fitness function, and a probability of selection, p_s is assigned to each member according to the following formula:

$$p_s(S_i) = \frac{f(S_i)}{F(P_t)} \qquad ...(2.5)$$

Individuals are then selected by generating a uniformly distributed number z between 0 and 1 and selecting the individual S_i such that $\sum_{k=1}^{i-1} p_s(S_k) < z \leq \sum_{k=1}^{i} p_s(S_k)$.

This method of selection is called fitness-proportionate selection, or roulette wheel selection (see Subsection 2.4.3a.) on page 24). This method is illustrated by the example given in Figure 6.

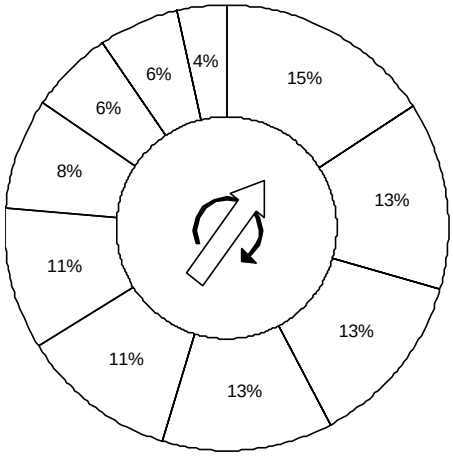


Figure 6: An Example of Fitness Proportionate Selection

The selection is repeated *n* (the population size) times. Multiple selections of the same individuals are allowed. A new population is formed containing only members of the previous generation, but the members are now expected to occur at a frequency proportional to their relative fitness.

Now the representation of a schema, *S*, in the population, P_t, can be measured by the number of individuals in P_t, belonging to *S*. This is denoted by $\xi(S, P_t)$. Further, it is useful to measure the relative fitness of the schema using the mean fitness of all the instances of *S* in P_t. This is denoted by $f(S, P_t)$. If selection alone determined the next generation, the expected value of $\xi(S, P_{t+1})$ could be calculated using the following formula:

$$\xi(S, P_{t+1}) = \frac{\xi(S, P_t) \cdot f(S, P_t)}{F(P_t)} \cdot n = \frac{\xi(S, P_t) \cdot f(S, P_t)}{\overline{F}(P_t)} \quad \text{...(2.6)}$$

Since $\xi(S, P_t) \cdot f(S, P_t)$ represents the total fitness of all instances of *S* at time *t*, it follows that $\frac{\xi(S, P_t) \cdot f(S, P_t)}{F(P_t)}$ represents the total proportion of the population fitness occupied by instances of *S*, *i.e.* the probability that an instance of *S* is selected each time a selection is made. Here *n* represents the population size and hence the number of selections that are made. The formula is simplified by observing that $\frac{F(P_t)}{n} = \overline{F}(P_t)$.

If only selection is performed, the fitter schemata (and individuals) will quickly dominate the population. However, no new solutions are generated. The existing population is simply repeatedly evaluated. In order to generate new candidates and yet maintain diversity in the

population, two other operations, crossover and mutation (see 2.4.1b.) on page 21), are used. They are controlled by the following parameters:

- p_c, the crossover rate, which determines the probability that an individual will participate in crossover during a single generation, and
- p_m, the mutation rate, which determines the probability that any particular bit in the population will be inverted.

For crossover, a group of individuals is selected from the population. Each individual has a probability p_c of being selected. If an uneven number of individuals is selected, one more individual should be added or removed at random to ensure there is an even number of individuals.

These individuals are then grouped randomly in pairs. The paring can be done in the order that the individuals were selected, since it is already random. For each pair a crossover location is selected and crossover is performed.

Although crossover provides a means by which new schemata are generated in the population, crossover can also disrupt existing schemata.

If an instance of a schema *S*, is selected for crossover and the crossover point occurs between the first and last defined bits of *S*, the schema could be disrupted and then neither offspring would be instances of S. The probability that a crossover will not occur between the outer defining bits of a schema is given by:

$$p'_{cd}(S) = 1 - p_c \cdot \frac{\Delta(S)}{m-1} \qquad ...(2.7)$$

Here m represents the length of the genome and m-1 represents the number of locations at which crossover can occur.

However, P_{cd}, does not accurately represent the probability that *S* will survive, since *S* cannot be disrupted by crossover if both parents are instances of *S*. If both parents are not instances of *S*, the schema could still remain intact, even if crossover does occur between defined bits. For example if S=(□101□10□) and S_i=(110101101) is an instance of *S*, but S_j=(001010101) is not, crossover after locus 5 would yield the offspring S_k=(110100101) and S_l=(001011101) and S would survive in S_k. So we amend equation 2.7 with:

$$p_{cd}(S) \leq 1 - p_c \cdot \frac{\Delta(S)}{m-1} \qquad ...(2.8)$$

After crossover, mutation is applied on a bit-by-bit basis to every gene in the population. Every gene has a p_m probability of undergoing mutation, *i.e.* of being inverted. Holland (1975, pp. 111) proposed that mutation be introduced as a background operator to ensure that particular alleles are never lost from a population. He argued that the role of mutation is not to generate new structures, as this role is already effectively filled by crossover. Thus mutation plays a purely disruptive role in the propagation of existing schemata, but serves as an "insurance policy" against the loss of alleles by ensuring that a GA, potentially, always has the full range of alleles to work with. The probability that a schema, *S*, will not be disrupted by mutation is given by the formula:

$$p_{md}(S) = (1 - p_m)^{o(S)}, \quad \text{... (2.9)}$$

because the mutation of any one of the defined bits is *S*, of which there are $o(S)$, would disrupt *S*. For $p_m \ll 1$, p_{md} can be approximated using the following formula[2]:

$$p_{md}(S) \approx 1 - p_m \cdot o(S) \quad \text{... (2.10)}$$

A combination of formulae 2.4, 2.6 and 2.8, results in:

$$\begin{aligned} \xi(S, P_{t+1}) &\geq \frac{\xi(S, P_t) \cdot f(S, P_t)}{\bar{F}(P_t)} \cdot (1 - p_c \cdot \frac{\Delta(S)}{m-1}) \cdot (1 - p_m \cdot o(S)) \\ &\geq \frac{\xi(S, P_t) \cdot f(S, P_t)}{\bar{F}(P_t)} \cdot (1 - p_c \cdot \frac{\Delta(S)}{m-1} - p_m \cdot o(S)) \end{aligned} \quad \text{...(2.11)}$$

This result is known as the Schemata Theorem and can be expressed in words (Michalewicz, 1992, p. 53):

c.) The Schemata Theorem

The schemata theorem is stated as follows:

Short, low-order, above average schemata receive exponentially increasing instances in subsequent generations of a GA.

All schemata will tend to increase or decrease exponentially, since their representation in the next generation is linearly proportional to their representation in the current generation. Short, low order, above average schemata will tend to have a proportionality constant greater than one,

---

[2] From the binomial theorem, we know $(1-b)^n = \sum_{i=0}^{n} \binom{n}{i} (-b)^i$.

Now if $b \ll 1$, then we consider b^i, where i>1, to be neglegible, and only the first two terms of the sum are significant. This yields $(1-b)^n \approx 1 - b \cdot n$.

while longer, higher order and all below average schemata will have proportionality constants less than one. Schemata will only continue their exponential growth as long as they remain above average.

Since all the schemata present in the population are sampled and processed simultaneously, the GA can process many more schemata than the individuals present in the population. This characteristic is called "implicit parallelism".

It is important to realize that the GA does not evaluate the average fitness of a schema, but only estimates it by sampling from the instances of the schema present in the population. Another interesting observation of the schemata theorem, is that a GA can successfully identify short, low-order schemata, but not necessarily complete solutions, which would be represented by long, high-order schemata. Since GAs can be successful at finding complete solutions, the building block hypothesis has been proposed by Goldberg (1989a) as a rough explanation of how GAs operate.

d.) The Building Block Hypothesis

The building block hypothesis states that

A genetic algorithm seeks near-optimal performance through the juxtaposition of short, low-order, high performance schemata, called building blocks.

So it is proposed that GAs operate by finding good building blocks, using mutation, crossover and selection and then combining them, using crossover, to produce complete solutions. There are shortcomings to this hypothesis, since not all problems will necessarily have solutions that can be represented by such building blocks. Such problems occur when building blocks that yield good improvements in isolation, yield poor performance when they are combined. This phenomenon is called deception and will be discussed in the following subsection.

e.) Epistasis and Deception

Epistasis in genetics refers to the effect that one gene has on another. The presence of an epistatic gene leads to suppression of the effect of another on a different locus (Reeves & Wright 1995). In GAs, epistasis is considered to be the effect genes have on the fitness contributions of one another.

	Fitness	
Genome	**No Epistasis**	**Epistasis**
00	0	0
01	3	3
10	1	1
11	4	6

Table 4: An Example of Epistasis

Consider a simple genome consisting of two genes represented in Table 4. If no epistasis is present, the fitness contribution of any gene is independent of any other gene. The fitness contribution of a gene is defined in a particular genome as the difference in the fitness of the genome, resulting from the presence of the particular gene.

In the case where no epistasis is present, it can be seen from Table 4 that the fitness contribution of each individual gene is independent of other genes. In this example, the contribution of the first gene is 3 and that of the second gene is 1.

In the case where there is no epistasis, the fitness of the genome where both genes are active is equal to the base fitness of zero plus the fitness contributions of all the active genes, in this case 0+3+1=4. If the fitness of a genome differs from the sum of the contributions of all the active genes contained in it, like in the second case, epistasis is present. In this case, there is a difference of 2 between the fitness contribution of one gene if the other changes. The size of the differences is a measure of the degree of epistasis that is present.

In this example only two-way interaction is considered. Multiple genes can interact to produce increasingly complex results. In GAs epistasis is associated with non-linearity, since any function that does not exhibit epistasis is simply determined by adding the individual fitness contribution of each gene to the fitness of the zero genome, $f(00...00)$. Such a function can be optimized feasibly by simply determining the fitness contributions for each gene experimentally and then setting each gene to its optimal allele.

Epistasis is often called synergism or suppression, depending on the meaning associated with the genes and the context of the problem.

By examining the way GAs construct solutions and understanding the epistasis present in the representation of solutions, attempts can be made to predict what problems are suited to be solved by using GAs, and what representations should be used for potential solutions. Considering the schemata theorem and the building block hypothesis, some implicit requirements on a problem's structure can be deduced that are necessary for it to produce building blocks:

The Building Block Requirements

1. Strong epistasis should exist within building blocks contributes. The loss of a single gene in a building block can destroy its contribution to a genome's fitness. Strong interaction between genes within a building block does not necessarily make a problem easier to solve by GAs, but it could give the GA an advantage.

2. Combining building blocks should improve the performance of an individual. If two good building blocks are combined, the resulting fitness should not be less than any one of the original building blocks.

3. Epistasis should be strong between nearby loci. This is necessary to ensure that building blocks are of low order and short length to prevent disruption by the genetic operators.

Mitchell, Forrest & Holland (1992) devised a class of functions, called Royal Road functions, to capture the essence of building blocks and to investigate the ability of GAs to optimize them. A Royal Road function assigns fitness to an individual via the presence of a predefined set of n schemata S. Each schema has an associated fitness coefficient c_i; the fitness of an individual x is defined as:

$$f(x) = \sum_{i=1}^{n} c \cdot b(x, s_i) \qquad \text{...(2.12)}$$

where $b(x,s_i) = 1$, if $x \in s_i$

and, $b(x,s_i) = 0$, if $x \notin s_i$

Since the addition of a schema s_i can only improve an individual, the combination of two fit schemata will have a fitness of at least the sum of the fitness of the individual schemata. Forrest & Mitchell (1993) tested GAs on a Royal Road function with a simple set of predefined building blocks. With such a set the optimal solution would contain all the blocks. A GA was compared to various hill-climbing approaches to optimise the Royal Road functions. Surprisingly, a hill-climbing algorithm with random initial solutions was the most successful. When the propagation of the building blocks in the GA was investigated, the reason for its poor performance was determined and could be explained as follows:

As more and more building blocks are found, they are combined into highly fit individuals. These individuals effectively retard the construction of the undiscovered building blocks by propagating their own unfit genes. Unfit genes get propagated on the performance of the already discovered building blocks. New building blocks that are discovered on the loci of these unfit genes are at a disadvantage in the selection process due to the higher fitness of the genomes associated with unfit genes. Such unfit genes are called "hitchhikers".

Although the fitness function of the Royal Road seems ideal for GAs, they suffer from hitchhiking because of a lack of diversity in the population.

The term "deception" was introduced by Goldberg (1987) to describe problems where the fitness functions of lower order schemata contain misleading information on higher order schemata. This violates the second building block requirement, so that combining two fit lower order schemata produces a less fit higher order schema. A fitness function where the optimal solution contains none of the winning lower order schemata is termed "fully deceptive".

For example, consider a function where the fitness of an individual is equal to the number of 1's in its representation, except for the individual with the representation (00...00), which has a fitness of m+1, where m is the length of the genome. Then all genomes other than this one will carry misleading information on the optimal solution.

Although it would seem that deceptive problems should be more difficult for GAs to solve than problems with no deception, Greffenstette (1993) argued that deception is neither a sufficient nor a necessary condition for a problem to be difficult to solve using a GA.

Although GAs were originally designed as adaptive systems that can continuously adjust to a changing environment (de Jongh 1993), they are often used as function optimisers and have proven to be effective in a wide range of applications.

GAs have been extended and hybridised to various degrees and have proven to be well suited to many different implementations. In this project, GAs will be used as a population-based component when developing a solution method for the sequential ordering problem.

2.5 PARTICLE SWARM OPTIMIZATION

Particle swarm optimisation is a recent development in population-based heuristics and will be used as a basis for the development of heuristics to solve the sequential ordering problem in this project.

The concept of particle swarm optimisation or PSO was developed by Eberhart and Kennedy (1995). This method embeds potential solutions to an optimisation problem as a population of particles in a search space. The position of a particle represents a candidate solution. The method optimises the problem by directing the movement of these solution particles towards promising positions previously identified by the population's movement with the objective of finding more promising positions.

One of the key strengths of this method is that it is very simple to implement and can be adapted to a wide variety of problems. Where no specialised algorithms are available to address an optimisation problem, PSO can be implemented speedily and it often yields acceptable results.

2.5.1 Origins of the particle swarm optimiser

The particle swarm optimiser was inspired by the flocking behaviour observed in birds and was later adapted to emulate the swarming behaviour around a food source (Eberhart & Kennedy, 1995).

The particle swarm optimiser attempts to find optimal solutions by influencing the movement of the particles in a manner that mimics the movements of insects in a swarm. Each particle is driven partially by its own knowledge of the environment and partially by observed behaviour of other members of the swarm.

This method attempts to strike a balance between exploiting public knowledge within the population through social interactions amongst particles and exploiting the knowledge gained by individual particles as they move around the search space.

2.5.2 The mechanics of a particle swarm optimiser

The PSO evaluates the velocities and movements of each particle in the swarm in time increments. During each time increment, each particle is evaluated, an adjustment is made to reflect its velocity, and its position is updated using the updated velocity.

A particle's velocity is influenced by the following factors:

- The particle's current velocity is used as a basis, this means that the particle maintains some **momentum** from the previous time increment. The degree to which momentum

is preserved is controlled by an inertia weight I, which is used to scale the particle's current velocity.

- An adjustment is made to the particle's velocity that is called the "cognitive" part of the velocity and represents the knowledge that the particle has about its environment. The particle's velocity is adjusted towards the best position that the particle previously occupied. The size of the adjustment is determined by the product of a parameter α and some uniformly distributed random number between 0 and 1.
- Another adjustment is made to the particle's velocity that is called the "social" part of the velocity and represents the knowledge that the entire population has about the environment. The particle's velocity is adjusted towards the best previous position that any particle in the swarm has. The size of the adjustment is determined by the product of a parameter β and some uniformly distributed random number between 0 and 1.

In this way a particle is drawn towards known good solutions by "cognitive" and "social" interaction. These behaviours will tend to exploit already known regions of the search space to identify promising solutions.

The momentum of a particle may cause it to overshoot the known solutions and will provide a mechanism that enables the particle to explore new regions of the search area. Shi and Eberhart (1998) explored the role of momentum in the PSO and found that without momentum, particles will tend to cluster around known local optima, thereby increasing the risk that the algorithm will get stuck without finding a global optimum. Velocity clamping is sometimes used to ensure that a particles momentum does not cause a velocity explosion. Velocity clamping is a mechanism by which the magnitude of a particle's velocity is limited to a fixed maximum.

By adjusting the parameters of the PSO, its behaviour can be adjusted between exploratory and exploitative. In general, larger parameter values will tend to cause more exploratory behaviour. Smaller parameter values, in particular for I and β is likely to lead to more exploitative behaviour.

The minimum requirements for applying PSO to an optimisation problem are:

- Parameters $I \geq 0$, $\alpha \geq 0$ and $\beta \geq 0$.
- An encoding to represent potential solutions as positions in a search space.
- A measurement function m that maps each position to the objective function value of the solution represented by the position.

- A definition of the concept of a velocity on the search space along with the following four operations (these abstract operators were introduced by Clerc (2000) to explain how PSO can be used to solve discrete optimisation problems):
  - Velocity scaling – coefficient $\otimes$ velocity $\rightarrow$ velocity
  - Velocity addition – velocity o velocity $\rightarrow$ velocity
  - Positional difference – position Θ position $\rightarrow$ velocity
  - Positional displacement – position $\oplus$ velocity $\rightarrow$ position

If these conditions are satisfied, we can apply the PSO algorithm as given below.

Pseudo-code for the particle swarm optimisation algorithm (for a minimisation problem)

```
 1: gbestfitness ← ∞
 2: for each particle in population do
 3:    location_i ← random location
 4:    velocity_i ← zero velocity
 5:    pbestlocation_i ← location_i
 6:    pbestfitness_i ← fitness(location_i)
 7: next particle
 8: while maximum iterations or minimum error criteria are not met do
 9:    for each particle in population do
10:       particlefitness ← fitness(location_i)
11:       if particlefitness < pbestfitness_i then
12:          pbestlocation_i ← location_i
13:          pbestfitness_i ← particlefitness
14:       end if
15:       if  pbestfitness_i < gbestfitness then
16:          gbestlocation ← pbestlocation_i
17:          gbestfitness ← pbestfitness_i
18:       end if
19:    next particle
20:    for each particle in population do
21:       velocity_i ← (I ⊗ velocity_i) o
                       ((α·U(0,1)⊗(pbestlocation_i Θ location_i)) o
                       (β·U(0,1)⊗( gbestlocation Θ location_i)))
22:       location_i ← location_i ⊕ velocity_i
23:    next particle
24: end while
25: return gbestlocation
```

2.5.3 Solving discrete problems using PSO

The most straightforward application of PSO would be when a particle's position and velocity are represented by vectors of real numbers. The particle operators could then be implemented through vector addition, subtraction and scaling. This implementation allows PSO to be used in a great number of continuous optimisation problems such as –

- Training artificial neural networks (Salerno 1997, Eberhart & Shi 1998, El-Gallad *et al.* 2001, Gudise & Venayagamoorthy 2003, Ismail & Engelbrecht 1999; Ismail & Engelbrecht 2000, Mendes *et al.* 2002)
- Designing electrical control systems (Fukuyama *et al.* 1999, Abido 2002, Conradie, Miikkulainen, & Aldrich 2002)
- Optimising conditions for fermentation processes (Cockshott & Hartman 2001)
- Model selection for forecasting applications (Voss & Feng 2002)
- Optimising CNC end-milling (El-Mounayri, Dulga & Deng 2003, Tandon & Kishawy 2002)

The application of PSO to optimisation problems was extended to discrete problems first by its designers (Kennedy & Eberhart 1997) when it was adapted to manipulate particles that were represented by binary strings, and later by Mohan & Al-Kazemi (2001) to manipulate sequences and solve permutation problems.

The key to implementing discrete problems is finding a usable definition of distance and direction in the discrete search space that allows the implementation of the four operations required to implement a PSO.

In the case of a sequencing problem, Mohan & Al-Kazemi (2001) described the method to be followed as –

- A particle position is represented by a sequence of a fixed length.
- A movement of velocity is represented by two components. First by a series of pair-wise exchanges of the elements in a sequence that changes one sequence to another. Second, by representing magnitude of the movement by an integer that determines how many of the exchanges should be performed. If the integer is greater than the number of exchanges in the sequence, the sequence of exchanges is repeated and if the integer is less than zero, the exchanges are applied in reverse order.
- If a velocity is scaled by a real number, the integer indicating the magnitude is multiplied by the scalar and the result is rounded to the adjacent integer with the largest absolute magnitude.

These definitions of position, velocity and scaling allow the implementation of a PSO that operates in a discrete search space by manipulating sequences.

Discrete implementations of the PSO have been used to solve a variety of combinatorial optimisation problems such as:

- Task assignment problems (Salman, Ahmad & Al-Madani 2002)
- Surveillance mission planning (Secrest 2001)
- Integer programming problems (Laskari, Parsopoulos & Vrahatis 2002)

Particle swarm optimisation is a relatively new method that at present is still the focus of much research. The method has shown promise in being flexible in terms of the problems it can be applied to and in being simple to implement. In this project this method will be used in combination with other heuristics, to construct more sophisticated search algorithms for the sequential ordering problem.

2.6 NEIGHBOURHOOD SEARCH AND LOCAL OPTIMA

In this project neighbourhood search methods are used to enhance the performance of "pure" population-based methods.

The combination of population-based and neighbourhood search methods are often used in so-called memetic algorithms. The term "memetic algorithm" is used to describe population-based methods that use all available knowledge of the problem under investigation (Mocato & Cotta 2002). Unlike generic population-based solvers, like GAs and PSO, memetic algorithms are tailored to exploit the specific characteristics of the problem they are designed to solve. One method for making such problem-specific knowledge available is to apply a neighbourhood search method that employs such knowledge to improve the solutions found by means of a population-based method.

A local search algorithm is defined by three elements:

1. a neighbourhood relation between solutions in the search space,
2. an objective function that guides the search, and
3. a search strategy that determines the criteria for adopting new solutions.

2.6.1 Neighbourhood relations

Neighbourhood search algorithms rely on the definition of a neighbourhood relation between solutions in a search space. A neighbourhood relation associates every potential solution, x, with a set of other potential solutions, $N(x)$. This set is called the neighbourhood of x.

Neighbourhood search algorithms seek to improve solutions by repeatedly evaluating the neighbourhood of the current solution and then replacing the current solution by one of its neighbours.

For a neighbourhood relation to be used effectively in a search, there needs to be an efficient method to generate the neighbours of a particular solution. For example, it might seem to be a good idea to define the neighbourhood of a solution as all the solutions in the search space that have a better objective function value than the current solution. However, for most problems, it would become very difficult to generate new neighbours once a very good, but non-optimal solution has been found.

Often, neighbourhood relations use the concept of distance to associate solutions that are in some sense near to each other.

The following examples of neighbourhood relations show how neighbourhoods can be constructed on different solution spaces.

The Euclidian distance between two vectors can be used to define a neighbourhood relation. The neighbourhood of a vector $\boldsymbol{x}$ can be defined as all vectors (in the same vector space) where the Euclidian distance to $\boldsymbol{x}$ is less than (or equal to) some fixed radius.

The Hamming distance between two binary strings of equal length can be used to define a neighbourhood relation. This hamming distance is the number of bits at which the vectors differ. For example, a neighbourhood of x can be defined as all the solutions where the hamming distance from x is equal to one. This method also allows all the neighbours to be generated by consecutively inverting the bits of x.

Pair-wise exchange between elements in a sequence can be used to define a neighbourhood relation that associates all sequences that can be transformed to one-another by means of such an exchange.

2-opt exchange is an often-used neighbourhood definition that associates all sequences that can be transformed to one-another by the juxtaposition of two adjacent elements.

For any solution space, any number of neighbourhood relations can be defined. The effectiveness of a neighbourhood search method is highly dependent on the neighbourhood relation that the method uses. Trade-offs that have to be considered when deciding on a neighbourhood relation include the size of the resulting neighbourhoods, the effort required to generate neighbouring solutions and the risk of becoming trapped in local optima (Michalewicz & Fogel 2002, p.42).

2.6.2 Local Optima

A local optimum under a given neighbourhood relation is a solution that has no neighbours with a better objective function value than itself. Once a local search has reached a local optimum, no improvements can be found by moving to one of the neighbours.

Multiple local optima present a challenge to local search methods due to the risk of becoming trapped at a local optimum and so being unable to progress to the global optimum.

Many search methods require an improved solution to continue and will stop once a local optimum has been reached. If such a method is deterministic in the sense that a starting solution will guarantee that the search method will end up on the same local optimum every time the search is performed, then the set of all the solutions that lead to that optimum, is called the "basin of attraction" of the local optimum (Michalewicz & Fogel 2002, p.43).

To prevent a search method from becoming trapped in local optima, some mechanism is required to allow for movement through less desirable solutions to escape the basins of attraction of local optima.

2.6.3 Neighbourhood search strategies

Once a neighbourhood relation has been chosen for the search, there are many search strategies that can be used to pick neighbours and explore the search space. Three typical methods that will be reviewed in this section are: hill climbing, simulated annealing and tabu search.

Simulated annealing and tabu search have some exploratory capabilities that are not required in the solvers developed as part of this project. However, they are included in this section to demonstrate how the same neighbourhood relations can be used to implement different search methods.

a.) Hill Climbing

Hill climbing is one of the most basic neighbourhood search methods. The method starts out with one possible solution and evaluates the neighbourhood to select another solution with a better objective function value. It then repeats the process with the new solution.

Hill climbing terminates when no better solutions can be found in the current solution's neighbourhood, *i.e.* when a local optimum is reached.

When the method consistently chooses the best solution in the neighbourhood, it is called "steepest-ascent" or "steepest-descent" hill climbing, depending on whether the problem is a maximisation or a minimisation problem.

b.) Simulated annealing

Simulated annealing is a method developed by Kirkpatrick, Gelatt and Vecchi (1983), that was inspired by the phenomenon of annealing in metallurgy. Metallurgical annealing changes the properties of metal by controlled heating and cooling of a metal object. The annealing of metal allows atoms in the metal to move around freely at high temperatures and as the temperature is reduced, atoms settle into configurations with lower internal energy than they originally started with.

When simulated annealing is used in local search, a temperature parameter T is used in the search. T starts at a high value and is reduced through an annealing schedule, typically to zero, as the search continues. Simulated annealing repeatedly selects candidates from the neighbourhood of the current solution and then decides whether or not to accept the new solution, based on the

objective function value and the current temperature of the system. If a new solution is accepted, it replaces the current solution and the process is repeated.

Many options are available for deciding whether or not to accept a solution. The following probability function is often used:

$$p_{accept} = \frac{1}{1+e^{\frac{f(s_{current})-f(s_{candidate})}{T}}} \quad \text{... 2.13}$$

where f is the objective function of a maximisation problem, $s_{current}$ is the current solution and $s_{candidate}$ is the candidate solution. If the problem is a minimisation problem, the positions of $s_{current}$ and $s_{candidate}$ in the equation are reversed.

As $T \to \infty$, this formula will result in 50% acceptance of any solution, regardless of the objective function value. When $T=0$, the method will accept any improved solution and reject all weaker solutions. For positive values of T, the probability of acceptance is always 0.5 if the solutions are the same, but the probability of acceptance of better solutions as well as the probability of rejection of poorer solutions increases as T decreases.

As the search method starts with a high value of T, there will be little discrimination between strong and weak solutions. As T decreases, the search will start to favour better solutions and should drift towards more promising regions in the search area. When T reaches 0, the search will only accept improvements and will perform like a hill climbing algorithm.

The length of a simulated annealing search is typically limited by the number of iterations required to reach $T=0$ and then some additional time to allow it to reach a local optimum. The search can also be terminated if no improvements are found within a set number of iterations.

Simulated annealing overcomes the tendency of local search algorithms to become trapped by local optima, by allowing the process to move against improving solutions while T is greater than 0.

One of the drawbacks of this implementation of simulated annealing, is that it becomes inefficient at accepting new solutions as $T \to 0$ due to the difficulty of finding better solutions when the current solution is already a good one.

In theory, simulated annealing is guaranteed to converge to a global optimum, provided that the search space is finite and the annealing schedule is infinitely long. Although this theoretical result is not achievable in practice, simulated annealing has proved to be very successful at solving actual sequencing problems.

c.) Tabu search

The tabu search algorithm was introduced by Glover & Knox (1998). Rather than being a specific implementation of a search method, tabu search specifies principles and mechanisms that can be used to construct a search strategy, specific to the problem under investigation. Such strategies are called "meta-heuristic" strategies.

The main concept behind the tabu search is that it has a "memory" which forces the search to avoid previously explored areas in the solution space. As with simulated annealing, tabu search was designed to prevent the search from becoming trapped at local optima, but unlike simulated annealing, tabu search is usually implemented as a deterministic method.

This subsection will briefly review some of the principles used in tabu search and present an example to demonstrate how it can be implemented.

The tabu search works in a manner similar to an ordinary hill climbing search method, but prevents entrapment in local optima by providing a mechanism that enables non-improving movements away from a local optimum and prevents movement that will cause the algorithm to "back track" and revisit old solutions. Tabu search uses three mechanisms to achieve this behaviour, namely:

- A *recency-based* memory used to record the transformations that the search uses in progressing from one solution to another. The reverse transformations that are required to restore solutions to their predecessors are stored in a so-called *tabu list* for a set number of iterations. As long as a transformation remains on the *tabu list*, it is forbidden and cannot be used during the search.
- An *aspiration criterion* used to prevent a *recency-based* memory from excluding exceptional solutions from the search. When an *aspiration criterion* is used, all solutions are evaluated, and solutions that are prohibited by the *recency-based* memory are allowed, provided they present a significant improvement to the current solution. The required level of improvement is specified by the *aspiration criterion*.
- A *frequency-based memory* used to encourage exploration of the search space by penalising often-used transformations. A *frequency-based memory* uses a longer period than the *recency-based* memory and instead of prohibiting specific transformations, the *frequency-based memory* will penalise often-used transformations. Depending on the implementation, the frequency based penalty can be applied across the board, or only when the search is forced to move towards less desirable solutions.

These mechanisms can be amended and implemented to suit the problem at hand. A tabu search containing all of these mechanisms can be implemented on an encoding of fixed length binary strings as the following illustration will explain.

Consider a maximisation problem with a solution space that consists of binary strings of length 10 and a neighbourhood relation that defines two strings as neighbours if they differ at only one location. Any transformation from one neighbour to another would be achieved by inverting a single bit.

- A *recency-based memory* could be implemented by recording which bits were inverted, and prohibit these bits from being inverted for a set number of iterations. To prevent the search from running out of alternatives, the number of iterations during which the prohibition remains in force would have to be less than the length of the string.

  Assume a string of length 10 where the memory remains effective for 5 iterations. If with a string x_0: 1101100011 and after one iteration, the search moves to x_2:1111100011 by inverting the third bit, the *recency-based memory* will prevent the third bit from being flipped again within the next 5 iterations.

- An *aspiration criterion* can be implemented by assigning the minimum improvement that is required to ignore prohibitions set by the *recency-based memory*.

  Assume an aspiration criterion that requires an improvement of 5 units to override the *recency-based memory*. If the string x_2:1111100011 has progressed to x_3:1111100111, and the objective function value $f(x_3) = 5$, then the search is allowed to proceed to x_4:1101100111, provided that x_4 is the best neighbour of x_3 and $f(x_4) \geq 10$; otherwise, the *recency-based memory* will prevent the inversion of the third bit.

- A *frequency-based memory* could be implemented by recording the number of bit inversions of each position of the solution string for the last 50 iterations. The objective function value of a solution that is reached by a particular inversion will be penalised by a multiple of the frequency of the bit that is inverted.

  Assume that the search has reached the fiftieth iteration and x_{50}: 0010011111, and that a 0.1 penalty is associated with the frequency of transitions. The frequency of each bit inversion is recorded in the following string:

Bit	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}
Frequency	3	4	4	3	5	1	4	9	9	8

If the search then considers the string s: 00000011111 for the next iterations, the *frequency-based memory* will apply a $0.1\times\ b_3 = 0.4$ penalty to the objective function of s when determining the next solution.

Tabu search has proven to be remarkably effective at addressing a wide variety of combinatorial optimisation problems.

As a meta-heuristic, tabu search allows a wide variety of implementations and can be tailored to specific problems. Since the original tabu search suggested by Glover and Knox (1998), many variations have been introduced and effectively implemented for sequencing problems.

Tabu search is a good example of how local search methods can be used effectively, even in the presence of multiple local optima.

In this section, the basic structure of local search algorithms was reviewed along with the role that neighbourhood relations and search strategies play in these heuristics. This project will focus on the ability of local search algorithms to effectively identify local optima and as such steepest-decent hill-climbing heuristics will be used.

2.7 CHAPTER SUMMARY

In this chapter the literature that relates to this project was introduced.

The first section introduces the sequential ordering problem, the specific problem that is being addressed in this project. This sequential ordering problem is a problem that is often encountered in the practice of industrial engineering and it is a difficult problem to solve optimally. Examples of the sequential ordering problem include the scheduling of a pick-up and delivery run for a delivery vehicle and machine scheduling.

The second section reviews the theory of computational complexity. Using complexity theory, computational problems can be classified according to how difficult they are to solve. The sequential ordering problem is identified as a member of the NPO-complete class of problems. Currently, there are no known methods to optimally solve optimisation problems that are NP-complete problems within a practical time period and problem solvers are required to use heuristics that can give good answers to these problems in acceptable time periods.

Sections 2.3, 2.4 and 2.5 review heuristic methods that can be used to address difficult optimisation problems. Specifically, in sections three and four two population-based heuristics are reviewed and in section five local search heuristics are reviewed. The combination of population-based and local search heuristics is a key focus of this project.

Genetic algorithms are reviewed in Section 2.3. These algorithms mimic the process of evolution in nature. This is achieved by maintaining a population of candidate solutions and simulating the exchange of solution properties through a process analogous to sexual reproduction and then by simulating natural selection by eliminating less desirable elements of the population. The theory supporting genetic algorithms as well as developments made throughout the history of genetic algorithms is covered in detail.

In Section 2.4, particle swarm optimisation is reviewed. Particle swarm optimisation is a relatively new population-based approach to solving both continuous and combinatorial optimisation problems. Like genetic algorithms, particle swarm optimisation also maintains a population of candidate solutions. However, unlike genetic algorithms, particle swarm optimisation does not use an evolutionary mechanism, but rather simulates the movements of a flock of bird or a swarm of insects to introduce changes to its population. This method uses a form of memory that drives the movement of particles toward areas in the solution space that are known to be promising.

Local search heuristics are reviewed in Section 2.5. Local search heuristics start with one solution and progressively improves this solution be making small incremental changes. These changes transform a solution into a neighbouring solution. One key difference between different local search heuristics is the definition of a solution's neighbours, or alternatively, what incremental changes are allowed to improve the solution. Local search heuristics do not maintain a population like the heuristics discussed in Sections 2.3 and 2.4, but focus on improving one solution at a time.

The following chapter defines a framework for developing a new heuristic that combines population-based and local search heuristics into a two-tiered architecture that aims to improve on the performance of its individual components.

3 A FRAMEWORK FOR A TWO TIER POPULATION-BASED SOLVER

The tools and techniques reviewed in Chapter 2 can be employed to construct a wide variety of algorithms for solving optimisation problems. This chapter describes a framework that will be used to construct the specific kind of solvers that are investigated in this project.

The framework that is proposed in this chapter is generic. Specific implementations will be investigated in the following chapter.

The first section will review the objectives of the solver. This will include performance criteria against which this kind of solver should be measured, as well as behavioural criteria that identify the properties that are to be investigated in this project.

The second section will explain the operating principles with which the two-tiered solvers must comply. When constructing a solver, these principles will be used as a crosscheck to ensure that the solver follows the desired strategy when searching for solutions to the sequential ordering problem.

The third section will specify the structure of the population-based solver in terms of the components that make up the solver and the particular roles and behaviour required of these components. The structural specification is checked against the operating principles specified in section two to demonstrate that the structure adheres to the desired operational strategy.

The fourth and final section of this chapter will provide a set of specific techniques and benchmarks against which the effectiveness of solvers, as described by this framework, can be measured.

3.1 OBJECTIVES OF THE TWO TIER POPULATION-BASED SOLVER

The solvers developed in this project are designed to meet two kinds of objectives.

First, it is recognised that many methods are available to solve specific problems and, to be of any practical value, a new method has to be competitive in providing acceptable solutions within a reasonable timeframe. Specific performance objectives will be described in this section. Section 3.4 on page 62 will discuss the measurement of the various methods against these performance objectives in more detail.

Second, as a research objective, this project will investigate specific behaviour that is exhibited by the two-tier population-based solvers described in this chapter. The behaviour that needs to be exhibited by these solvers will be specified in this section.

3.1.1 Performance objectives for two-tier population-based solvers

The primary function of any optimiser is to give an optimal or near optimal solution to a specific instance of a problem as quickly as possible. Accordingly, the primary function of the solvers in this project is to find a good solution to a sequential ordering problem within a time period that is at least comparable to other methods, if not an improvement. The following criteria will be used to evaluate the solvers:

a.) Computational complexity

Since the solvers under investigation do not specifically construct solutions, but rather iteratively explore the search space, it is not meaningful to talk about the time required to execute the algorithm. More significantly, the complexity of executing a single iteration will be analysed to understand how quickly the solver can scan the search space.

The computational complexity of executing a single iteration can, as defined in Chapter 2, be expressed in terms of the size of the problem being solved.

The computational complexity of solving the sequential ordering problem can be expressed as a function of the length of the sequence required by the sequential ordering problem, or by the number of constraints imposed on the sequential ordering problem. In this project, the computational complexity will be expressed as a function of the required sequence length, since

1. search methods are expected to be more sensitive to the sequence length and,
2. the number of constraints are small relative to, and limited by, the sequence length.

b.) The ability to find global optima

Since the solvers under investigation are heuristic in nature, there is no guarantee that a global optimum can be achieved for any particular problem. Also, if a random problem is given to the solver, it will not be easy to determine if a solution found by the solver is in fact a global optimum. To measure the ability of a particular solver to find global optima, the following approach will be used:

First, the performance of the solver needs to be measured against benchmark problems for which the objective function value of a global optimum is known in advance. This will allow the analyst to know when the solver has found such a solution and stop the search.

Second, the time will be measured which a solver requires to reach the global optimum for a particular problem. This test will only be practical if the benchmark problem is small enough to be solved within a reasonable time span.

This experiment can be repeated a number of times, using the same problem and the same time limit, and the proportion of successful trials can be used as a basis for comparing different solvers.

c.) The quality of solutions generated in a limited time span

Heuristic methods, such as the solvers being developed in this project, sacrifice guaranteed performance in order to rapidly deliver acceptable solutions. To measure the ability of a given solver to achieve this objective, the following approach is used:

First a benchmark problem is to be identified. Each solver is then allowed a limited period of time to produce a candidate solution. The quality of the candidate solution is used as a measure to compare different solvers. A number of trials can be generated for each solver using this method and the resulting solutions are used to compare the performance of the solvers on the particular problem.

The performance of different two-tiered solvers will be measured to compare the effectiveness of different implementations. However, to gauge the usefulness of using the newly developed solvers, these methods will also be tested against selected other solvers that are known to be efficient in solving the sequencing problem in practice.

3.1.2 Behavioural objectives of the two-tiered solvers

The behavioural objectives of these solvers form the research basis for this project.

The solvers are developed using two tiers to isolate exploratory and exploitative behaviour in the heuristic and to allow an investigation of the interaction between these two behavioural components.

First the performance of these behavioural components in isolation can be determined by deactivating alternate tiers of the solver. The performance of the complete solver can then be used to determine how co-operation between tiers contribute to the solver's performance.

Second, by comparing solvers with different methods of information exchange between tiers, the contribution of the information exchange mechanisms can be measured to estimate the trade-off between the computational cost of information exchange and the performance advantage gained.

a.) Exploration vs. exploitation

The two-tier population-based solver framework seeks to separate exploratory behaviour that seeks to identify promising new regions of the search space and exploitative behaviour that seeks to improve existing solutions.

The first tier is the exploratory tier that maintains a population of candidate solutions. The function of this population-based tier is to prevent stagnation by continually seeking new, promising regions of the search space.

The second tier is the exploitative or local search tier. This tier operates on one solution at a time and seeks to improve existing solutions by making small incremental changes. It uses local search methods to achieve this objective.

This structure allows the analysis of the tiers in isolation. The exploratory tier can operate as a stand-alone solver and the local search tier can be operated by repeatedly improving randomly generated candidate solutions. The effectiveness of the tiers in isolation can then be compared to each other and to the full solver in terms of the performance objectives that were identified in the previous subsection.

b.) Information exchange

There are many options for exchanging information between the two tiers of a solver. The simplest mechanism would be to feed information from the population-based tier to the local search tier only, and more complex mechanisms feed information in both directions.

The more complex and abundant information exchange enabled by the two-tier approach can yield improvements in the effectiveness of a search, but at the cost of a higher computational load.

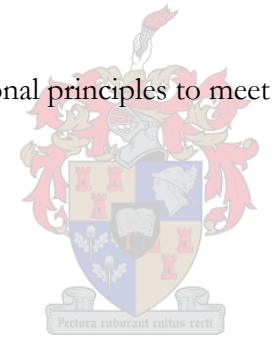
By using the same basic implementations of the two tiers in the solver, but by changing the scope of the information exchange between them, different performances may be observed. Such differential performance will be investigated to evaluate the effectiveness of information exchange in different situations.

This section reviewed two objectives for the development of the two-tiered solvers:

The first objective is performance driven to determine if the relevant solvers are of practical use and to provide a basis for comparing these solvers to each other as well as to other kinds of solvers.

The second objective specified the behaviour required from the two-tiered solvers. This objective will ensure that specific behaviour can be isolated and its contribution to the overall effectiveness of the solvers can be measured.

The next section specifies operational principles to meet these behavioural objectives.



3.2 THE OPERATING PRINCIPLES OF THE TWO-TIERED SOLVER

This section explains how the two-tiered solvers function. The section will review the principles that drive these solvers and show how they ensure that the behavioural objective set in the previous section, will be achieved.

The following principles guide the overall behaviour of the solvers:

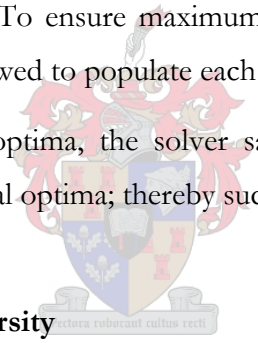
- local optima have to be identified and populated,
- population diversity has to be promoted,
- the average population performance has to be improved, and
- functional overlap between tiers has to be minimised.

3.2.1 Identify and populate local optima

The two-tiered solvers will follow the "populate-and-conquer" principle found in some GA implementations (Winston 1993, pp. 505-528).

The first tier of the solver is population-based and each member of the population should be driven towards a local optimum. To ensure maximum coverage of the local optima, only one population member should be allowed to populate each local optimum.

By deliberately populating local optima, the solver sacrifices members of the population by allowing them to be trapped in local optima; thereby such local optima are marked as risks for the remainder of the population.



3.2.2 Promote population diversity

To support the first principle, the solver will seek to promote diversity in the population. This means that similar members of the population will tend to be driven apart, thereby preventing more than one solution from occupying the same local optimum.

The first requirement for promoting diversity is to find a way of measuring it. A means of defining distance between members of the population and some mechanism to credit population members for being different to their neighbours, are required.

Promoting diversity usually comes at the cost of the performance of individual population members. A balance must be kept between maintaining diversity in the population and ensuring that population members can pursue improvements that might bring them closer to their neighbours.

3.2.3 Improve average population performance

Since the number of local optima can exceed the number of members of a population, the solver cannot stop the search when all the population members have reached different local optima.

The solver has to keep seeking new local optima to replace the worst performing candidates in its population. The average performance of the individuals in the population has to be driven to improve until the population represents the best set of local optima in the search space.

In addition to guiding the search in the direction of the local optima, this principle also drives the population towards containing a number of good, but significantly different candidate solutions to the problem being solved. This gives a decision maker reasonable options when the formal objective function and constraints are not the only considerations.

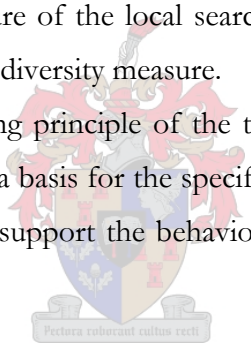
3.2.4 Minimise functional overlap between tiers

Finally, since one behavioural objective is to explicitly separate the behaviour of the tiers, the solver should be implemented to minimise the functional overlap between tiers.

Functional overlap may occur if:

- the local search tier exhibits exploratory behaviour or the ability to escape local optima,
- the population-based tier performs local optimisation similar to the local search tier, or
- the neighbourhood structure of the local search tier's local search method is related to the population-based tier's diversity measure.

This section explained the operating principle of the two-tiered solvers that are investigated in this project. These principles form a basis for the specific structure of such solvers, which will be described in the next section, and support the behavioural objectives contained in the previous section.



3.3 THE STRUCTURE OF THE TWO-TIERED SOLVER

The two-tiered solver contains a population-based tier; a local search tier as well as an information exchange mechanism between these tiers. These three components need to work together to support the operating principles set out in the previous section.

This section specifies the roles and behaviour of each tier in the complete solver and gives direction on how information exchange could be implemented between the tiers.

3.3.1 The population-based tier

The population-based tier has three roles in the solver:

- it maintains knowledge from one iteration to the next,
- it fulfils the exploratory function of the solver and
- it applies problem independent techniques, *i.e.* it avoids undue exploitation of problem specific knowledge.

The population-based tier **maintains knowledge** between iterations by maintaining multiple candidate solutions. As the solver explores the search space and identifies other promising regions in the search space, the population of candidate solutions is updated to reflect the new knowledge.

Following the "populate-and-conquer" principle explained in the previous section, each candidate solution in the population can be associated with a local optimum and the *basin of attraction* (see Subsection 2.6.2 on page 43) that surrounds this optimum. New solutions generated during the search can be evaluated, on the local search tier level, to determine whether they fall within a new region of the search space, or one already occupied by an existing solution. Feedback to the first, population-based tier is then used to reflect the new knowledge in the population.

The population-based tier fulfils its **exploratory function** by the following means:

It generates new solutions either by combining and extending existing solutions in the search space, or by randomly adding new solutions. The process through which the new solutions are generated uses stochastic inputs and is unpredictable to ensure that this tier has the freedom to explore outside localised constraints and performance slopes.

The exploratory behaviour is partially driven by a diversity objective to ensure that population members are credited for being different. This diversity objective ensures that population members can sacrifice performance for diversity without being excessively penalised. The diversity objective is measured by using some form of distance to measure how close population

members are to one another. Members that are distant from other population members are then credited for contributing to the diversity of the population.

Finally, the population-based tier **avoids using problem specific knowledge** to solve a problem. This means that the population-based tier could be used to solve any form of the sequential ordering problem and does not assume special properties like symmetry, metric objective functions or constraint depths (see Subsection 2.2.3 on page 11) when performing its search.

3.3.2 The local search tier

The local search tier is implemented as a local search algorithm with some restrictions to ensure that it fulfils its roles. This subsection will review the function of the local search tier in the overall solver.

The local search tier fulfils the following roles within the solver:

- it exploits newly generated solutions to identify local optima,
- it defines predictable and stable basins of attraction for local optima, and
- it uses problem specific knowledge to improve the solver's performance.

The primary purpose of a local search algorithm and by implication, that the local search tier is to improve an existing solution. The local search tier of the solver exists to take solutions from the first, population-based tier, and to incrementally improve them until a local optimum is reached.

One key characteristic of the local search tier is that it does not employ stochastic variables and has repeatable performance. This means that the search will always drive a particular candidate solution to the same local optimum. In this way, the local search tier implicitly defines the basin of attraction of each local optimum by associating every solution in the search space with the particular local optimum of the basin in which it resides.

The local search tier is also allowed to exploit problem specific properties to enhance the performance of the solver. For instance, the local search tier could be specialised to search problems with symmetry, a metric objective function or specific constraint structures.

The local search tier is not concerned with escaping local optima as is the case with some local search implementations (see Subsection 2.6.3 on page 44). The local search tier is meant to employ a fast and greedy neighbourhood search to predictably associate all candidate solutions with local optima.

An important parameter in the design of the local search tier is the definition of the neighbourhood structure. The neighbourhood structure determines which solutions are local optima. The neighbourhood structure also contributes to the efficiency of the search through the

number of neighbours that can be generated for any particular solution and the number of transitions that are required to reach local optima.

3.3.3 Information exchange between tiers

The final element of the two-tiered solver is constituted of an exchange of information between the two tiers, where selected methods are employed to enable such information exchange. Different approaches to information exchange between tiers can result in substantial differences in computational load on the solver, which have to be traded off against improvements in performance. This subsection reviews the methods that can be employed to exchange information between tiers in order to improve the overall performance of the solver.

One element of information exchange that must always be present, is the communication of new solutions from the population-based tier to the local search tier. The latter must always evaluate new solutions generated by the population-based tier, after which feedback may flow back to the population-based tier.

The following kinds of communication can be used to feed information from the local search tier back to the first:

- **No feedback** is a possibility. In this case, the population-based tier will operate independently of the local search tier. Using this method, the local search tier is unable to contribute to the average performance of the population-based tier and the population-based tier will be unable to identify solutions that fall within the basin of attraction of the same local optimum. This method violates key principles as identified in Section 3.2 on page 56 and is not acceptable as a communication method for the two-tiered solver.
- **Fitness feedback** occurs when the optimised objective value function of a new candidate solution of the local search tier, is communicated back to the population-based tier. Although this method allows the population-based tier to determine whether a candidate solution falls within a promising region of the search space, it also does not allow the population-based tier to identify solutions that fall within the basin of attraction of the same local optimum. Although this method may improve the performance of the solver, it does not allow the population-based tier to fulfil its function of ensuring diversity within the population after the local search tier has processed it. This method therefore is also not acceptable as a communication method for the two-tiered solver.
- **Replacement** is implemented when the optimised solution from the local search tier replaces the original solution in the population-based tier. This will ensure that, after the local search tier has processed information regarding the population, all candidate

solutions in the population will be local optima. This method enables the population-based tier to ensure diversity by identifying solutions that reside in the basin of attraction of the same local optima, since they will all be returned from the local search tier as the same local optimum. The diversity objective in the population-based tier should then ensure that duplicate solutions are eliminated to make way for new candidate solutions. This method is a simple and acceptable communication method between the two tiers of the two-tiered solver.

- **Full feedback** occurs when the locally optimal solution found by a local search tier is made accessible to the population-based tier for making decisions, but does not replace the solution found in the population-based tier. Using this method, the population-based tier will keep the population that it generated before the local search tier was engaged, but it will know which local optimum each solution is associated with and what the objective function value of that local optimum is. To make full use of this method, the population-based tier has to be explicitly designed to make use of this information. This method is more sophisticated than a pure replacement strategy and the performance benefit needs to justify the additional computational effort. This method is an acceptable communication method for the two-tiered solver.

Figure 7 shows the structure of the two-tiered solver described in this subsection. Candidate solutions are passed from the population-based tier to the local search tier. The local search tier improves candidate solutions and returns the results to the population-based tier. The population-based tier then adjusts the population and the process is repeated.

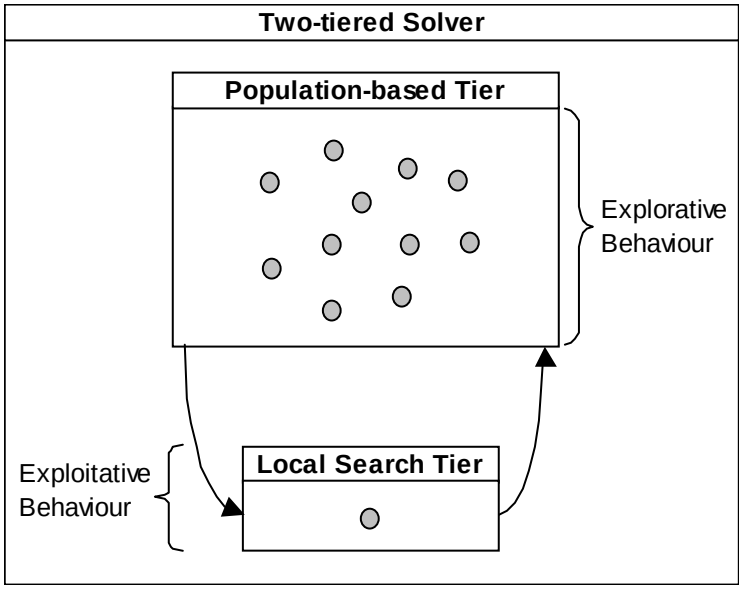


Figure 7: The Structure of a Generic Two-tiered Solver

3.4 MEASURING THE PERFORMANCE OF THE TWO-TIERED SOLVER

In Section 3.1 on page 52, the performance objectives of the two-tiered solver were introduced. In this section the means that will be employed to measure solver performance in this project will be described. In particular the following questions will be addressed:

- What benchmark problems will be used to measure performance?
- How is a solver's ability to reach a global optimum measured?
- How is a solver's ability to rapidly find acceptable solutions measured?
- What benefit does the two-tiered structure bring to the solver?
- How does the solver compare to other methods when solving the sequential ordering problem?

In this section the methods that will be used to measure and compare two-tiered solvers will be specified. In addition to comparing different implementations to each other, two other methods that are known to perform well on the sequential ordering problem will also be evaluated in order to compare the various methods meaningfully.

3.4.1 Benchmark problems

The solvers need to be measured against problems that present some degree of challenge and that will allow the solvers to be compared against other methods. Some requirements of such problems are:

- They have to be known and accepted as a sufficiently challenging test set.
- The optimal solutions for the problems have to be known in advance to enable the search to stop once a global optimum is identified.
- The test set has to include problems of different sizes to test the change in performance as the size of the problem varies.
- Ideally, the test set should contain problems derived from real-world applications.

The TSPLIB is a library of combinatorial problems related to the travelling salesman problem, which contains 41 instances of the sequential ordering problem. This library has been chosen as a source for the benchmark problem set for this project. TSPLIB was compiled by Gerhard Reinelt of the University of Heidelberg. It can be accessed via the internet at http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/ (accessed: August 2006).

Ten problems were selected for evaluation. The selection was made to cover a range of problems with sizes of less than or equal to 100 locations and to include all the available symmetric instances of the sequential ordering problem in this range.

Due to the size of the test instances, these problems will not be reproduced in this document.

3.4.2 Measuring the ability to identify global optima

A key performance criterion for any heuristic designed to solve a NPO-hard optimisation problem is its ability to identify global optima. This subsection will explain how the ability of a solver to identify global optima will be evaluated.

There are at least two difficulties that face any method that attempts to find the global optimum of a NPO-hard optimisation problem:

1. There is no way of distinguishing between locally optimal solutions and globally optimal solutions. Without information to distinguish between locally and globally optimal solutions, a solver could become trapped in locally optimal solutions, which are globally suboptimal.
2. The sheer number of options that result from the complexity of the problem may prevent the solver from identifying a globally optimal solution within an acceptable time period.

The ability of a solver to identify global optima can be measured by evaluating the ability of the solver to overcome these difficulties. By using benchmarks where the objective function value of an optimal solution is known, the evaluator of the solver is able to distinguish between locally and globally optimal solutions. Thereby the evaluator can measure the ability of the solver to identify the global optima.

A solver's ability to efficiently identify global optima will be estimated by running the solver on a problem until a global optimum is identified. The time that is required for the solver to identify the optimum is used to quantify the ability of the solver to efficiently identify the optimum. Since the required time is a stochastic variable, several trials are required for each solver on each problem to allow a meaningful comparison between solvers.

Due to practical constraints, the time available for a trial will be limited. For comparison, the performance of any trial that fails to achieve a global optimum will be considered to be worse than any trial that did achieve a global optimum and equivalent to all other trials that failed to achieve a global optimum.

3.4.3 Measuring the time required to achieve acceptable solutions

Often, especially when larger problems are considered, the ability of the solver to achieve a certain level of performance quickly is more important than to find an absolute global optimum. This subsection will explain how a solver's ability to produce good solutions within a limited time period is measured.

The performance of a solver on a particular problem can be estimated by allowing the solvers to evaluate the problem over a limited, predetermined time period and considering the objective function value of the resulting solution.

Since the result is a stochastic variable, a number of trials are performed for each solver on each problem and the results are used to compare different solvers to one another.

3.4.4 Measuring the contribution of different tiers to solver performance

The contribution of each tier to the total solver's performance can be measured by measuring the different tiers individually and comparing the results to the performance of the complete solver.

The measurement methods described in Subsections 3.4.2 and 3.4.3 on pages 63 and 64 will then be used to evaluate solvers and compare them to one another.

This subsection will explain how individual tiers will be evaluated when they do not form part of a two-tiered solver.

The population-based tier can be executed independently by ensuring that the local search tier does not generate any new information. This is done by passing solutions that would be evaluated and processed by the local search tier straight back to the population-based tier without alterations. Effectively, this would be equivalent to completely eliminating the local search tier and would ensure that any form of information exchange implemented between tiers can be accommodated without compromise to the functioning of tier one on its own (See subsection 3.3.3 on page 60).

The local search tier can be independently evaluated by feeding randomly generated solutions to the local search tier one by one. This would destroy the ability of the solver to maintain knowledge through the population-based tier's population.

If any one of the tiers is removed, the information exchange between the tiers is clearly also removed.

3.4.5 Comparing two-tiered solvers to competing methods

For a solver to be used in actual (rather than only academic) problem solving, its performance has to be comparable to other methods that address the same problem. For this project two other methods have been selected for such comparison based on their reported performance and their widespread use in actual practice:

1. The first method is an implementation of the simulated annealing algorithm. Simulated annealing type solvers are often used in practice to solve a variety of sequencing problems. The specific implementation of this problem is described in Chapter 4.

2. The second method is a specialised genetic algorithm called edge assembly crossover, developed by Nagata and Kobayashi (1997). This method was developed for solving the travelling salesman problems and succeeded in identifying global optima for 9 benchmark asymmetrical travelling salesman problems from the TSPLIB. The method will be implemented to solve the sequential ordering problem as will be described in detail in Chapter 4.

These methods were chosen arbitrarily to reflect one population-based method and one neighbourhood search based method. They do not represent best practice. Several problems exist when attempting to identify best practice benchmarks:

1. Many proprietary methods exist that may not be accessible for benchmarking purposes,
2. The field of combinatorial optimisation is under development and what may be regarded as best practice at one point in time is unlikely to remain so and
3. Until the question of whether or not P=NP is answered, the existence of a polynomial time algorithm that can solve NPO problems exactly cannot be discounted. Such an algorithm could, in theory, make heuristic methods obsolete.

This section described the methods that will be used to evaluate implementations of the two-tiered solver. These solvers need to be measured in their ability to identify global optima and their ability to deliver acceptable solutions within a practically acceptable time period.

3.5 CHAPTER SUMMARY

This chapter described the framework for developing two-tiered heuristic solvers for solving the sequential ordering problem. The framework combines a population-based component as well as a local search component that work together and aims to improve on their individual performances.

The first section stated the objectives of a two-tiered solver. The first set of objectives is performance based and requires that the solver is useful in practice and should be able to compete with alternative methods that are designed to solve the same problem. The second set of objectives sets behavioural requirements for the two tiers of the solver. Specifically, the population-based tier's behaviour must be exploratory and must ensure diversity in the population of candidate solutions. The local search tier's behaviour must be exploitative and must focus on small, "greedy" incremental improvements of existing solutions. The behavioural requirements are necessary to quantify the benefit of combining these two, seemingly conflicting, behaviours into a single solver.

The second section identified four operating principles that support the objectives identified in Section 3.1. The first principle is that the two-tiered solver should identify and populate local optima. The second principle is the maintenance of population diversity. By populating local optima and ensuring diversity, the solver aims to maintain a set of locally optimal, but significantly diverse solutions to a given problem. The next operating principle is to continuously seek to improve the population. The final principle, to support efficiency in the execution of the heuristic, is to limit the functional overlap between the two tiers of the solver.

Section 3.3 described the structure of the two-tiered solver. The roles of the two tiers were described and linked to the operating principles in Section 3.2. The following components and their roles were identified: The population-based tier maintains knowledge about the solution space through a set of candidate solutions, it fulfils the exploratory function of the solver and it applies techniques that are independent of problem specific characteristics. The local search tier exploits newly generated solutions to identify local optima, it implicitly defines predictable and stable basins of attraction for local optima in the search space and it can use problem specific knowledge to improve the performance of the two-tiered solver. One final component of a two-tiered solver is the means by which information is exchanged between tiers. Any two-tiered solver needs to pass the solutions from the population-based tier down to the local search tier, but different levels of feedback from the local search tier to the population-based can be accommodated in a two-tiered solver.

In the next chapter, the implementation of the two-tiered solvers is specified. The specific implementation of both local search and population-based tiers are specified and valid compositions of these tiers into full two-tiered solvers are identified. The two competing solution methods for the sequential ordering problem are also identified and specified in the following chapter.

4 TWO-TIERED SOLVERS FOR THE SEQUENTIAL ORDERING PROBLEM

The previous chapter described the framework for implementing a two-tiered solver. In this chapter the implementation of several variations of such solvers will be described in detail. The performance of these solvers will be evaluated against the criteria that were developed in the previous chapter. These evaluations will then be used as a basis to study the interaction between the separate tiers of such solvers and to compare the performance of the solvers to one another and to competing methods for solving the sequential ordering problem.

The first section in the chapter will provide a general overview of the way the solvers are implemented. This includes the data structures used to represent the problem, the way that interaction between the two-tiers of the solvers is facilitated, and the way that constraints are addressed in these implementations.

The second section provides detailed descriptions of the implementations of the population-based tiers of all the solvers that are being investigated in this project. This will include a detailed description of the program flow, the diversity measures that are used to prevent unwanted convergence and an analysis of the computational complexity of the implementations.

The third section explains the implementation of the local search methods used in the local search tier implementations of the two-tiered solvers. Besides detailed program-flow, this section will explain how these algorithms address constraints; it will review specific properties of the solvers and provide an analysis of the computational complexity of the specific implementations.

The fourth section lists the combinations of the first and local search tiers described in the second and third sections that will be combined and evaluated in this project. These implementations are selected to adhere to the design principles of the previous chapter.

In the fifth section the implementation of competing methods to solve the sequential ordering problem is described. These implementations are:

- An implementation of a simulated annealing heuristic for the sequential ordering problem.
- A GA that uses edge assembly crossover (EAX) (Nagata & Kobayashi 1997) to solve the sequential ordering problem.

The Java program code for all the solver implementations is available in soft copy.

4.1 AN OVERVIEW OF THE IMPLEMENTATION OF TWO-TIERED SOLVERS

This subsection contains an overview of how the solvers were implemented for this project.

All solvers and evaluation methods that were developed were implemented in the Java programming language. Java is an object-orientated programming language that allows the first and second level tier implementations to be developed separately and then to be combined afterwards by standardising the way that two tiers interact.

In the remainder of this section, the following questions will be answered:

- How is a specific instance of the sequential ordering problem represented?
- How are the population-based and local search tier implementations structured to ensure that they interact predictably?
- How do the two-tier population-based solvers resolve the precedence constraints of the sequential ordering problem?
- How are single tiers implemented so that they can be evaluated independently?

4.1.1 Representation of the sequential ordering problem

To represent the sequential ordering problem the following elements are required:

- a transition cost matrix that contains the costs of moving from one element to the next,
- a list of precedence constraints that specifies the order in which a particular pair of nodes must appear for a solution to be feasible, and
- the first and last nodes.

The assumption that the problem is a minimisation problem is implicit. Programmatically an instance of the sequential ordering problem is represented as a Java object with the following data structures:

- The cost matrix is represented by a two-dimensional array of real numbers that contains the transition costs between two nodes. The nodes are ordered so that the starting node is represented by the first index and the end node is represented by the final index. The examples in TSPLIB contain complete cost matrices, however if there is no link between two nodes, a very large number could be used represent the link and the analyst would have to verify that the final result is feasible.
- Precedent constraints are represented by a two-dimensional array of Boolean variables that are true when a precedence constraint exists between two nodes and false otherwise. The indexes for the cost and the constraint matrices are identical.

To ensure the proper representation of the problem, these two checks are automatically performed on the constraint matrix when a new problem is captured into this structure.

First the constraints are checked to ensure that the problem does not contain conflicting constraints. For example, if a problem requires specifically node *A* to be selected before node *B*, node *B* to be selected before node *C* and node *C* to be selected before node *A*, the problem would have no feasible solutions. Such problem instances are considered invalid and will not be considered here.

Second, the constraint matrix is checked to eliminate redundant constraints. For example, if node *A* is required before node *B* and node *B* is required before node *C*, then a constraint that node *A* is required before node *C* is redundant and will be disregarded when populating the constraint matrix. Although this check is not required to ensure the integrity of the problem, it will influence the way that constraints are taken into consideration.

4.1.2 Implementation of the population-based and local search tiers of the two-tiered solver

The population-based and local search tiers of the solvers can be implemented independently and combined as required. To make this possible, the individual tiers are implemented to fit a standard pattern that allows any population-based and local search tier implementations to interact without needing to consider the detailed implementation of the other tier. This characteristic is called encapsulation and is made possible through an object-orientated language like Java.

In this particular implementation of two-tiered solvers, the information exchange method between the tiers is implemented as part of the population-based tier structure. The local search tier will provide full feedback of improved solutions found by its local search mechanism, but the degree to which this information is used at the population-based tier is decided within its own implementation. If no feedback is required, the population-based tier will simply ignore the local search tier's feedback, but as more information is required, the population-based tier implementation will regulate the degree to which the feedback is used.

The local search tier will be implemented as simple, independent local search algorithms that receive candidate solutions from the population-based tier, apply local search heuristics, and return the locally optimised solutions. This tier will process solutions one by one. The local search tier implementations in this project all use a steepest-descent mechanism to traverse the neighbourhood, the only difference between implementations will be the neighbourhood relation that is used. This means that for each solution processed by a second-tier heuristic, the entire neighbourhood of this candidate must be examined to ensure the steepest descent.

4.1.3 Resolving precedence constraints

In this subsection, the way in which precedence constraints are resolved in this project is explained. This method is not the only way in which these constraints can be addressed.

For any infeasible solution, the degree by which a constraint is violated can be quantified. If a constraint is not violated, the degree of violation is zero. If a constraint is violated, the degree by which the constraint is violated is defined to be equal to the distance between the restricted nodes. For example if {*A,B,C,D,E,F*} is a candidate solution sequence and there is a constraint that requires *E* before *B*, then the degree of violation for this constraint is counted as 3. The degree of infeasibility of a candidate solution of a particular problem is defined as the sum of the degrees of violation for all the precedence constraints.

The degree of infeasibility provides a yardstick for how infeasible a solution may be and provides some mechanisms for driving infeasible solutions towards feasibility. To ensure that there is always an incentive to improve the feasibility of candidate solutions, the degree of infeasibility is always used with the objective function value to compare solutions. A solution with a lower degree of infeasibility is always regarded as superior to one with a higher degree of feasibility, regardless of the objective function value. This principle is very useful when candidate solutions need to be ranked. If candidate solutions have the same degree of infeasibility, then the candidate solution with the lower objective function will be regarded as superior. This will ensure that any comparison of candidate solutions in the population-based tier favour more feasible solutions. Local search tier implementations will also select neighbours that improve feasibility before seeking to improve the objective function value. Once feasible solutions are found, any search mechanisms that use the degree of infeasibility in this way will seek to improve the objective function value without compromising feasibility.

4.1.4 Diversity measures for population-based tier implementations

One of the principles introduced in Section 3.2 on page 56 was to promote diversity in the population of the population-based tier. This section will review several ways in which diversity can be measured. The key to a useful population diversity measure is a measure that meaningfully quantifies the distance between two candidate solutions. A population diversity measure will then be constructed by calculating the sum of the inverse squared distances between the candidate solutions in the population.

The first measure of diversity lies in counting the number of corresponding element positions between two solution sequences, excluding the first and the last elements. For example, if the second, third and fifth nodes in both sequences are the same, but no others except the first and

final nodes, the distance between the solutions will be three. This method does not recognise similarities between solutions that share subsequences if they do not occur on exactly the same locations in the solution sequence.

Another measure of distance would be to count the number of transitions shared between two solutions. This method will quickly recognise similar subsequences, but it places diversity in competition with performance by assigning short distances between solutions that share the least costly transitions.

Finally, neighbourhood structures can be used to determine distance. The distance between two candidate solutions can be defined as the minimum number of neighbourhood transitions required to transform from one solution to the other. This method often provides a good way to measure diversity, but it has at least two drawbacks. First, it could be computationally inefficient to determine the minimum number of neighbourhood transitions required, depending on the neighbourhood structure being used. Second, if the neighbourhood structure used for diversity measurements is the same structure used by the local search tier, then there is overlap between the functionality of the two tiers. Due to these drawbacks, this distance measure will not be used in this project.

4.1.5 Single tier implementations for comparison

Single tier implementations are used in this project to measure the benefit of combining the different behaviours made available by the two tiers. To exploit the implementation of the two-tiered structure for this purpose, a dummy instance of each tier will be implemented. This dummy tier contains minimal functionality and is combined with the tier being evaluated to produce complete solvers.

The first tier dummy is a random solution generator. It will randomly generate candidate solutions for the problem and feed them down to a functioning local search tier for processing. This first tier does not maintain a population like other population-based tier implementations and ignores feedback from the second tier. The results from the second tier are then used for evaluation.

The second tier dummy receives solutions from the population-based tier and passes them straight back without modification.

4.2 POPULATION-BASED TIER IMPLEMENTATIONS FOR TWO-TIERED SOLVERS

This section provides the specifications for three different population-based tier implementations used in two-tiered population-based solvers in this project.

The first implementation is based on particle swarm optimisation and uses a 2-opt neighbourhood structure to determine direction and distance between particles. This method uses full feedback from a local search tier to improve performance. Unlike a normal PSO (see Section 2.5 on page 37), which uses the best previous position of a particle to direct the search, this method uses the optimised position found by the local search tier, to direct the search. This method contains no additional mechanism to promote diversity.

The second implementation is a GA-based method that performs crossover through a 2-opt neighbourhood structure. This method replaces candidate solutions with feedback from the local search tier. Rank-space selection (see Section 2.4.3b.) on page 27) is used to ensure diversity in the population.

The third implementation is based on a traditional GA that uses partially mapped crossover (PMX) (Goldberg, 1989a). This method replaces candidate solutions with feedback from the local search tier. It uses rank-space selection to ensure diversity in the population.

The time complexity per solver iteration is determined for each implementation. For the time complexity analysis, the local search tier's contribution is not included. Since the number of local search-tier evaluations are related only to the population size, the complexity analysis can be done independently and the tier with the greatest time complexity will dominate and determine the overall time complexity of the combined solver.

4.2.1 Particle swarm optimisation using 2-opt neighbourhood transitions

This implementation uses the traditional particle swarm operators explored in Section 2.5 on page 37.

Particle positions are used to represent candidate solutions to a sequential ordering problem. Each position is represented by a sequence of transitions that represent a potential solution to the sequential ordering problem.

The displacement between two sequences is defined as the shortest series of pair-wise exchanges of adjacent elements that will transform one sequence to transform to another. Since more than one such sequence is possible, one of them will be generated arbitrarily as required.

Velocity is defined by a series of pair-wise exchanges of adjacent elements and an integer that indicates the number of exchanges to be performed. This is described as the magnitude of the velocity. If the magnitude exceeds the number of available exchanges, the sequence of exchanges will start to repeat. If the magnitude is negative, exchanges will be performed in reverse order.

After each iteration, the population is fed through the local search tier and the feedback from the local search tier is used to direct particle movement during the next iteration. A traditional particle swarm optimiser records the best position achieved by each particle to influence that particle's movement. This variation, however, uses the position returned by the locally optimising local search tier. The best position found by the search at any particular point in time influences the movement of all particles in the population. This position is also identified through the feedback from the local search tier.

The following operations are required to enable the calculation of particle positions and velocities:

- Velocity scaling – coefficient $\otimes$ velocity $\rightarrow$ velocity
  - Velocity scaling is achieved by multiplying the magnitude of a particle velocity by the scaling coefficient and rounding the result to the adjacent integer with the largest absolute value. The time required for this operation does not depend on the required sequence length and therefore it does not contribute to the time complexity of this implementation.
- Positional difference – position Θ position $\rightarrow$ velocity
  - The positional difference between two sequences indicates a velocity, which results in the first sequence if it is applied to the second sequence. The magnitude of the velocity is equal to the number of pair-wise exchanges of the

velocity. The following algorithm can be used to calculate the positional difference:

Pseudo-code to calculate positional differences between sequence1 and sequence2

```
 1: tempsequence ← sequence2
 2: velocityvector ← empty
 3: velocitymagnitude ← 0
 4: distancearray ← array of zero's with the length of sequence1
 5: repeat
 6:    totalgap ← 0
 7:    pivot ← (number of elements in tempsequence)/2
 8:    pivotdistance ←   absolute difference of the indices of
                         the pivot element in tempsequence and
                         sequence1
 9:    for each element in tempsequence
10:       tempdistance ← absolute difference of the indices of
                         this element in tempsequence and
                         sequence1
11:       totalgap ← totalgap + tempdistance
12:       if tempdistance > pivotdistance then
13:          pivot ← element
14:          pivotdistance ← tempdistance
15:       end if
16:    next element
17:    mindist ← ∞
18:    for each element in tempsequence
19:       if element <> pivot then
20:          testseq ← tempsequence
21:          tempval ← testseq [pivot]
22:          testsequence[pivot] ← testsequence[element]
23:          testsequence[element] ← tempval
24:          if mindist > distanceFunction[3](testseq,sequence2) then
25:             mindist ← distanceFunction(testseq,sequence2)
26:             candidateseq ← testseq
27:             sencondpivot ← element
28:          end if
29:       end if
30:    next element
31:    tempsequence ← candidateseq
32:    add (pivot, secondpivot) to velocityvector
33:    velocitymagnitude ← velocitymagnitude + 1
34: until tempsequence = sequence1
```

[3] The **distanceFunction** used in the algorithm returns the sum of the absolute distances between the positions of each element in the two sequences. An O(*n*) implementation of this function is described by Tagawa *et al.* (1998)

```
35: return velocityvector
```

The worst case performance of this algorithm will result in a velocity with magnitude O(l), where l is the length of the input sequences. Since every element in the sequence has to be evaluated for each exchange in the path, the time complexity of this code is O(l^2).

There are often several velocities that will transform the second sequence into the first. For this implementation, the velocity determined by the preceding method is used.

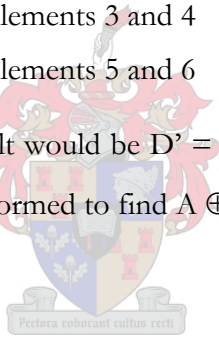
- Positional displacement – position $\oplus$ velocity $\rightarrow$ position
  - Positional displacement is achieved by applying the exchanges in the velocity sequence to the position vector. A number of exchanges equal to the magnitude of the velocity is applied. Negative magnitude will result in the exchanges being applied in reverse order. The time required for this operation does not depend on the required sequence length and therefore it does not contribute to the time complexity of this implementation.
- Velocity addition – velocity $\circ$ velocity $\rightarrow$ velocity
  - Velocity addition is achieved by starting with an arbitrary sequence and applying successive positional displacement of the two input velocities. The positional difference between the resulting sequence and the starting sequence is returned as the sum of the velocities. The only component of this operation that is dependent on the required sequence length is the calculation of a positional difference. Therefore this operation also has a complexity of O(l).

For example, given two sequences A=[2,4,1,7,3,6,5,8] and B=[1,2,3,4,5,6,7,8], a velocity between A and B, D, can be represented as a) a sequence of indices of elements to be exchanged with their immediate successors and b) a integer indicating the number of exchanges that are required: D = B ⊖ A = [4,2,5,6,1,3,5]-7.

Note that for the purposes of this example, the first element is considered to be the successor of the last element. If D is applied to A, then exactly 8 exchanges are applied in order, starting with the first index in D's index sequence. A ⊕ D = B can be verified as follows:

0. [2,4,1,7,3,6,5,8]
1. [2,4,1,3,7,6,5,8] – exchanged elements 4 and 5
2. [2,1,4,3,7,6,5,8] – exchanged elements 2 and 3
3. [2,1,4,3,6,7,5,8] – exchanged elements 5 and 6
4. [2,1,4,3,6,5,7,8] – exchanged elements 6 and 7
5. [1,2,4,3,6,5,7,8] – exchanged elements 1 and 2
6. [1,2,3,4,6,5,7,8] – exchanged elements 3 and 4
7. [1,2,3,4,6,5,7,8] – exchanged elements 5 and 6

If D was scaled by 0.5, then the result would be D' = 0.5⊗D = [4,2,5,6,1,3,5]-**4** and to calculate A ⊕ D', only four exchanges are performed to find A ⊕ D'=[2,1,4,3,6,5,7,8].



The following pseudo-code describes the program flow of this implementation:

Pseudo-code for particle swarm optimisation using 2-opt neighbourhood transitions

```
 1: gbestfitness ← ∞
 2: for each particle in population do
 3:    location_i ← random location
 4:    velocity_i ← zero velocity
 5:    pbestlocation_i ← location_i
 6:    pbestfitness_i ← fitness(location_i)
 7: next particle
 8: while maximum iterations or minimum error criteria are not met do
 9:    for each particle in population do
10:       optimisedlocation ← localSearchTierOptimise(location_i)
11:       particlefitness ← fitness(optimisedlocation)
12:       if particlefitness < pbestfitness_i then
13:          pbestlocation_i ← optimisedlocation
14:          pbestfitness_i ← particlefitness
15:       end if
16:       if  pbestfitness_i < gbestfitness then
17:          gbestlocation ← pbestlocation_i
18:          gbestfitness ← pbestfitness_i
19:       end if
20:    next particle
21:    for each particle in population do
22:       velocity_i ←  velocity_i ∘
                        ((α·U(0,1)⊗(pbestlocation_i⊖location_i))∘
                        (β·U(0,1)⊗( gbestlocation⊖location_i)))
23:       location_i ← location_i ⊕ velocity_i
24:    next particle
25: end while
30: return best gbestlocation
```

The factors that contribute to the time complexity of the algorithms are the evaluation of the objective function and the calculation of positional difference. Since evaluation is $O(l)$ and positional difference calculation is $O(l^2)$ and the operations are not nested, this algorithm has a time complexity of $O(l^2)$ per iteration.

This implementation accepts the population size, α and β as input parameters. Termination criteria will depend on the circumstances in which the solver is used. For instance, the runtime or number of iterations could be limited, a predetermined level of performance could be required or there could be a limit on the time or number of iterations that is allowed without an improvement.

4.2.2 GA-based solver using 2-opt inter- and extrapolation

This implementation uses the same positional and velocity representations used in the PSO implementation, but instead of applying the movement rules of a particle swarm, the velocity is used to facilitate the crossover operation in a GA.

This implementation uses four stages to update its population of candidate solutions:

1. **Generate offspring**: Using a crossover procedure, described later in this section, the GA produces candidate offspring. Each sequence in the population has a probability of p_c to be selected for crossover and parents are paired off at random. Parents remain candidates and are not discarded during this stage.

2. **Introduce mutation**: Each element in each sequence in the resulting population can be selected for mutation with a probability of p_m. When an element is selected for mutation, it is randomly relocated within its sequence.

3. **Local search tier feedback**: The new generation is optimised by the second-tier and the resulting candidates replace their predecessors in the population.

4. **Selection**: Candidates are selected from the new population using rank-space selection (see Subsection 2.4.3b.) on page 27) until the next generation reaches a predetermined population size. Each candidate can only appear once in the new generation. The number of corresponding nodes on two sequences is used to measure diversity

The core mechanism of this implementation is the crossover methods used to generate new offspring. When two population members are selected as parents, the offspring are generated by first generating a path between the two parents and then randomly selecting a solution that lies on this path. The path is defined by a series of pair-wise exchanges of adjacent elements in the sequence. This path is generated using the same method that is used to generate the positional differences between two sequences in the previous implementation (see Subsection 4.2.1 on page 73). This method of generating new candidate solutions is called interpolation, since it generates a candidate solution that lies between the two parents.

This interpolation method can be enhanced by allowing potential solutions that extend beyond the original path. This is achieved by either applying the exchanges in the path in reverse, similar to negative movement in the PSO implementation. The path can be extended in the other direction by allowing the repetition of the first part of the path after all the exchanges in the path have been applied. This is similar to movement in the previous implementation where the magnitude of the movement exceeds the path length. If a candidate solution is generated on the

extended path, the method is called extrapolation, since an offspring is outside the region between the two parent solutions, but still lies on the same path. Figure 8 illustrates conceptually how interpolation and extrapolation can be interpreted. The elements numbered 1 and 2 represent the parents and the element marked O represents the offspring. When interpolation is applied, the offspring lies between the parents. When extrapolation is applied, the offspring lies outside the area between the parents, but on the same path.

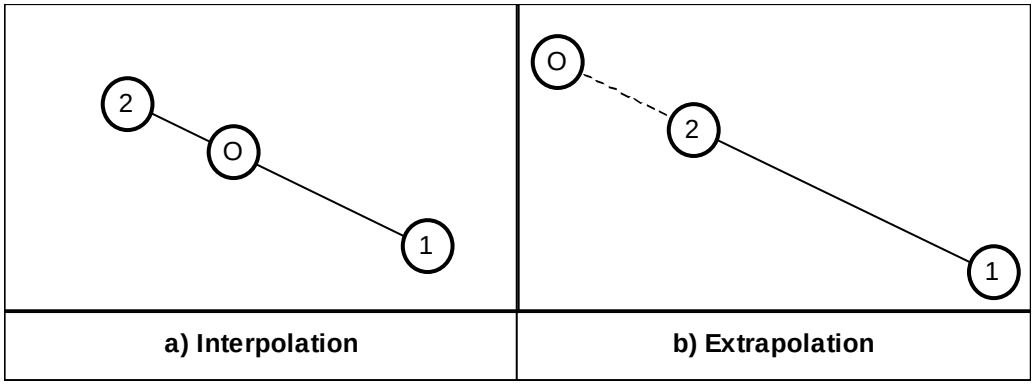


Figure 8: A Conceptual Example of Interpolation and Extrapolation

In this population-based tier implementation both interpolation and extrapolation are allowed. The magnitude of extrapolation is limited to half the distance between the parents.

The following pseudo-code describes the program-flow of this implementation:

Pseudo-code for the GA-based solver using 2-opt inter- and extrapolation

```
generate a random initial population
do
   parentset ← empty
   for each candidate in population do
      add candidate to parentset with probability p_c
   next candidate
   repeat
      randomly select parent1 and parent2 from parentset
      offspring ← inter-extrapolationCrossover(parent1,parent2)
      add offspring to population
      remove parent1 and parent2 from parentset
   until number of elements in parentset ≤ 0
   for each candidate in population do
      for each element in candidate do
         with probability p_m, randomly relocate element in candidate
      next element
   next candidate
   for each candidate in population do
      improvedcandidate ← localSearchTierOptimise(candidate)
      in population, replace candidate with improvedcandidate
   next candidate
   newpopulation ← empty
   repeat
      selectedCandidate ← rankSpaceSelection(population)
      add selectedCandidate to newpopulation
      remove selectedCandidate from population
   until the size of newpopulation = a predetermined popsize
   population ← newpopulation
until maximum iterations or minimum error criteria is met
return best candidate in population
```

The mutation operation scans every element in every sequence once, and then with a fixed probability reallocates a uniformly distributed random proportion of a sequence. This results in a time complexity of $O(l^2)$ for the mutation operation. The time requirements of rank-space selection is dependent on the population size and only the time required to evaluate individual solutions is a dependent on the actual problem size. Therefore rank-space selection has a time complexity of $O(l)$ if measured as a function of the problem size.

In this algorithm the most complex operations are the crossover procedure which uses the positional difference calculation and the mutation operation, both with time complexity $O(l^2)$.

Since these operations are not nested within any other operations that contribute to the time complexity of the algorithm, the time complexity of the algorithm is $O(l^2)$ per iteration.

This implementation accepts *popsize*, p_c and p_m as input parameters.

4.2.3 GA using partially mapped crossover (PMX)

This implementation of the first solver tier is almost identical to the second implementation in the previous section. The only difference between the two implementations is the crossover method used to generate new offspring. This method uses partially mapped crossover or PMX instead of inter- and extrapolation used by the previous GA.

PMX was used by Goldberg (1989a) to combine sequences for the travelling salesman problem. PMX creates a "child" or children that preserve a random subsequence found in one of the parents. It is implemented by selecting a random subsequence from one of the parents and then performing a series of pair-wise exchanges of elements in the second sequence to duplicate this subsequence. By swapping the role of the two parents, a second child can be generated. See Figure 9 for an example of PMX.

Each child is constructed by starting with one parent and then swapping each element in the selected subsequence with the corresponding element in the other parent. For example, Child 1 starts with a copy of Parent 2 and is constructed by then exchanged element 2 (3) with the element that is in position 2 in Parent 1 (10). In this case, element 2 is exchanged with element 8. This exchange procedure is repeated for elements 3 to 5, the remainder of the randomly selected subsequence.

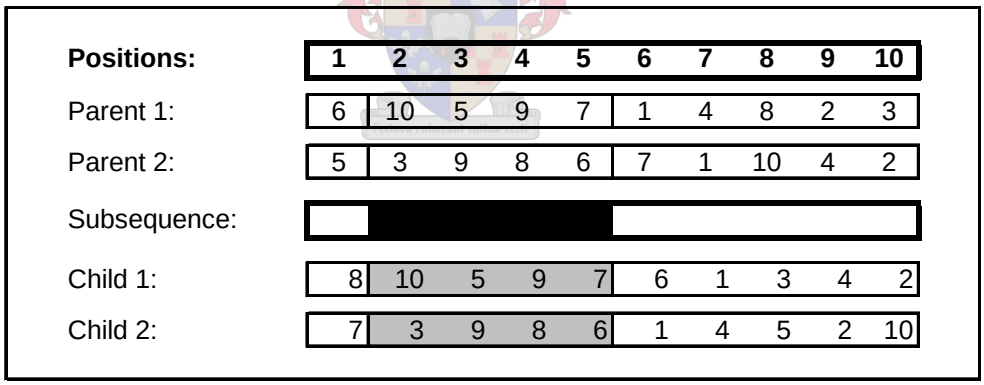


Figure 9: A Partially Mapped Crossover (PMX) Example

For this implementation, only one of the children will be used when performing PMX.

As in the previous GA implementation, the following phases are used to update the population from one generation to the next:

1. **Generate offspring**: Using the partially mapped crossover procedure, the GA produces candidate offspring. Each sequence in the population has a probability of p_c to be selected for crossover and parents are paired off at random. Parents remain candidates and are not discarded during this stage.

2. **Introduce mutation**: Each element in each sequence in the resulting population can be selected for mutation with a probability of p_m. When an element is selected for mutation, it is randomly relocated within its sequence.

3. **Local search tier feedback**: The new generation is optimised by the second-tier and the resulting candidates replace their predecessors in the population.

4. **Selection**: Candidates are selected from the new population using rank-space selection (see Subsection 2.4.3b.) on page 27) until the next generation reaches a predetermined population size. Each candidate can only appear once in the new generation.

The rank-space selection used in this implementation uses the number of corresponding nodes on two sequences to measure diversity.

The following pseudo-code describes the program-flow of this implementation:

Pseudo-code for the GA-based solver using partially mapped crossover

```
 1: generate a random initial population
 2: do
 3:    parentset ← empty
 4:    for each candidate in population do
 5:       add candidate to parentset with probability p_c
 6:    next candidate
 7:    repeat
 8:       randomly select parent1 and parent2 from parentset
 9:       offspring ← partiallyMappedCrossover(parent1,parent2)
10:       add offspring to population
11:       remove parent1 and parent2 from parentset
12:    until number of elements in parentset ≤ 0
13:    for each candidate in population do
14:       for each element in candidate do
15:          with probability p_m, randomly relocate element in candidate
16:       next element
17:    next candidate
18:    for each candidate in population do
19:       improvedcandidate ← localSearchTierOptimise(candidate)
20:       in population, replace candidate with improvedcandidate
21:    next candidate
22:    newpopulation ← empty
23:    repeat
24:       selectedCandidate ← rankSpaceSelection(population)
25:       add selectedCandidate to newpopulation
26:       remove selectedCandidate from population
27:    until the size of newpopulation = a predetermined popsize
28:    population ← newpopulation
29: until maximum iterations or minimum error criteria is met
30: return best candidate in population
```

In this algorithm, the crossover operation has a complexity of $O(l)$. Mutation dominates with a complexity $O(l^2)$ and the resulting time complexity for the algorithm is $O(l^2)$ per iteration.

This implementation accepts *popsize*, p_c and p_m as input parameters.

In this section, three implementations for the population-based tier of a two-tiered solver are specified. These implementations can be combined with any local search tier implementation to form a complete two-tiered solver. The next section will provide an overview of the local search tier implementations that are used in this project.

4.3 LOCAL SEARCH TIER IMPLEMENTATIONS FOR TWO-TIERED SOLVERS

This section specifies the implementations of the local search tiers of the solvers used in this project. The specific implementation and program flow of each search mechanism is introduced. Specific characteristics of each implementation are investigated to determine where it can take advantage of specific problem characteristics and the complexity of the implementations is quantified.

All of the local search tier implementations are variations of a steepest descent neighbourhood search. Each method starts with a candidate solution and evaluates the objective function value as well as the degree of infeasibility (see Subsection 4.1.3 on page 70) of every neighbour of this solution. The search then selects a neighbour from the set of neighbours with the lowest degree of infeasibility that has the lowest objective function value. The selected neighbour replaces the starting solution and the process is repeated until no more improvements are possible. The final solution is returned as the output of the search.

The following method is used to determine the degree of infeasibility of a solution:

Pseudo-code for calculating the degree of infeasibility of a potential solution

```
1: degreeofviolation ← 0
2: for each constraint in problem
3:    index1 ← index of firstconstraintelement in potentailsolution
4:    index2 ← index of secondconstraintelement in potentailsolution
5:    degreeofviolation ← degreeofviolation + max(0,index1 – index2)
6: next constraint
7: return degreeofviolation
```

The time required to calculate the degree of infeasibility is not a function of the length of the sequence and therefore this code does not contribute to the computational complexity of the overall solver when measured as a function of the sequence length.

The following generic program is used to implement the local search tier. The only difference between the different implementations is the set of neighbours that is generated:

Pseudo-code for local search tier neighbourhood implementations

```
 1: bestneighbour ← candidatesolution
 2: repeat
 3:    currentsolution ← bestneighbour
 4:    bestdegreeofviolation ← degreeOfViolation(currentsolution)
 5:    bestobjectivefunctionvalue ← objectiveFunctionValue(currentsolution)
 6:    for each neighbour of currentsolution (generated by neighbourhood relation)
 7:       degreeofviolation ← degreeOfViolation(neighbour)
 8:       if (degreeofviolation = bestdegreeofviolation) then
 9:          objectivefunctionvalue ← objectiveFunctionValue(neighbour)
10:          if (objectivefunctionvalue < bestobjectivefunctionvalue) then
11:             bestneighbour ← neighbour
12:             bestdegreeofviolation ← degreeOfViolation(bestneighbour)
13:             bestobjectivefunctionvalue ← objectiveFunctionValue(bestneighbour)
14:          end if
15:       else if (degreeofviolation < bestdegreeofviolation)
16:          bestneighbour ← neighbour
17:          bestdegreeofviolation ← degreeOfViolation(bestneighbour)
18:          bestobjectivefunctionvalue ← objectiveFunctionValue(bestneighbour)
19:       end if
20:    next neighbour
21: until currentsolution = bestneighbour
22: return currentsolution
```

Without the calculation of neighbours this algorithm is independent of the sequence length. The complexity of this algorithm is always determined by the time complexity, of the process that generates the full set of neighbours. These processes will be discussed individually in the following four subsections.

4.3.1 2-opt neighbourhood definition

The 2-opt neighbourhood relation generates neighbours by the pair-wise exchange of adjacent elements in a sequence. Pair-wise exchanges are also allowed between the first and last element of each sequence. This creates a neighbourhood relation where every sequence has exactly l neighbours, where l is the number of elements in the sequence.

Every neighbour of a given sequence can be generated by selecting a specific element in the sequence and exchanging the selected element's position with that of the next element. An example of how neighbours are generated using the 2-opt relation is shown in Figure 10.

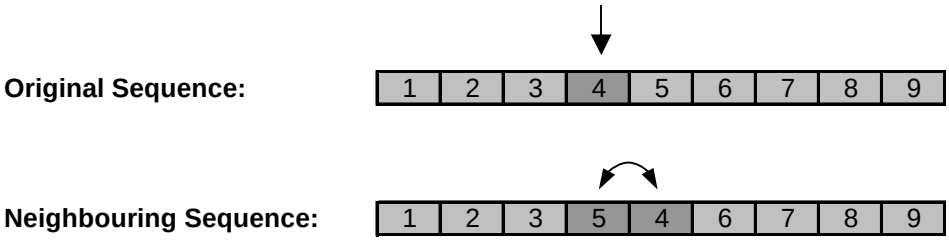


Figure 10: An Example of 2-opt Neighbour Generation

Since there are l possible neighbours and the time required to generate a neighbour is fixed, this method has a time complexity of $O(l)$ per iteration.

The 2-opt neighbourhood relation has the following properties:

1. Neighbours differ by a maximum of three transition cost elements (two if the cost structure is symmetric).
2. Only constraints associated with the two exchanged elements are affected.

The 2-opt neighbourhood is the smallest neighbourhood structure of the local search methods being investigated (l neighbours) and neighbours lie close to each other when using most forms of diversity measurement.

4.3.2 Substring inversion neighbourhood definition

Substring inversion generates neighbours by selecting a pair of nodes and inverting the sequence from one to the other (including the selected nodes).

This substring can wrap around the sequence if the first element occurs later in the sequence than the last. For example, if a sequence has 10 elements, then, if position 3 and then 6 are selected as the borders for the subsequence, the elements at positions 3,4,5 and 6 will be reversed and placed at positions 6,5,4 and 3 respectively. However, if position 6 is selected first and then position 3, the elements in positions 6,7,8,9,10,1,2 and 3 will be reversed and moved to positions 3,2,1,10,9,8,7 and 6 respectively. An example of how neighbours are generated using the substring inversion neighbourhood relation is shown in Figure 11.

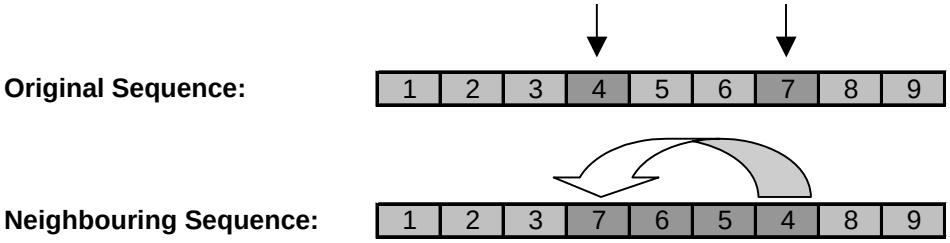


Figure 11: An Example of Substring Inversion Neighbour Generation

There are $l(l\text{-}1)$ possible ways to select a subsequence, but l of these combinations will give the same result: full reversal of the sequence. Therefore, this neighbourhood definition will generate $(l\text{-}1)/2$ unique neighbours and, on average, the cost for generating a new neighbour will be $O(l)$. This means that this method has a computational complexity of $O(l^3)$ per iteration.

The substring inversion neighbourhood relation has the following properties:

1. For symmetric cost functions, neighbours differ by a maximum of two transition cost elements.
2. The status of constraints where both elements lie on the inverted subsequence is reversed, i.e previously satisfied constraints become unsatisfied and previously unsatisfied constraints become satisfied.
3. All other constraints remain unaffected.

The substring inversion neighbourhood is a relatively large neighbourhood with $(l\text{-}1)^2$ unique neighbours for each candidate solution. This neighbourhood respects symmetry by making relatively small changes to the transition cost elements of a candidate solution if the sequential ordering problem is symmetric, while still having a large impact on the degree of infeasibility.

Since this method is computationally relatively expensive, its use is not recommended for asymmetric problems where changes to the cost structure are unpredictable.

4.3.3 Pair-wise node exchange neighbourhood definition

Pair-wise node exchange generates neighbours by swapping the position of two elements in a sequence. Each candidate solution has $l(l\text{-}1)/2$ possible neighbours and the time required to generate a neighbour is independent of the required sequence length. An example of how neighbours are generated using the pair-wise node exchange relation is shown in Figure 12.

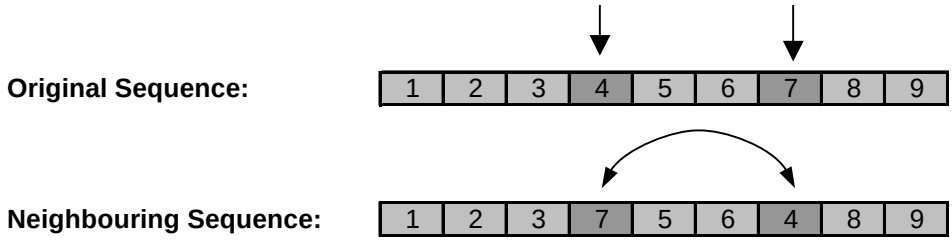


Figure 12: An Example of Pair-wise Node Exchange Neighbour Generation

This method can generate a set of neighbours with time complexity $O(l^2)$.

The pair-wise node exchange neighbourhood relation has the following properties:

1. Neighbours differ by a maximum of four transition cost elements, regardless of symmetry.
2. Only constraints associated with the exchanged nodes are affected.

The pair-wise node exchange is a relatively large neighbourhood that can be generated with medium time complexity. This method is insensitive to symmetry and is better suited to asymmetric problems than some of the other methods are. Neighbours using this relation are close to each other by most diversity measures.

4.3.4 Single node shift neighbourhood definition

Single node shift generates neighbours by displacing a single element in a sequence to any other location in the sequence. An example of this process is shown in Figure 13. Node 7 is selected as the node that will be shifted and it will be shifted to the location following node 4.

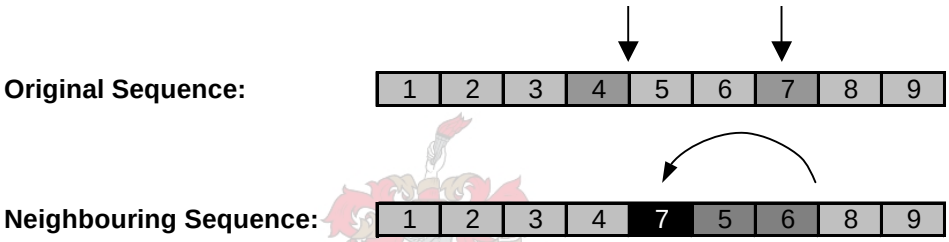


Figure 13: An Example of Single Node Shift Neighbour Generation

This method generates $l(l\text{-}1)$ different neighbours and, on average, the cost for generating a new neighbour will be $O(l)$. This means that this method has a computational complexity of $O(l^3)$ per iteration.

The single node shift neighbourhood relation has the following properties:

1. Neighbours differ by a maximum of four transition cost elements, regardless of symmetry.
2. Only constraints associated with the shifted node are affected.

Of all the neighbourhood relations that were discussed, this one typically has the smallest impact on constraints, since only one element is moved. Under this relation neighbours are close when measured with the shared transition method, but not necessarily when measured by corresponding element positions.

4.4 AN OVERVIEW OF COMPLETE IMPLEMENTATIONS OF TWO-TIERED SOLVERS

In this section the complete set of two-tiered solvers that are evaluated in this project are identified. Potentially every population-based tier implementation (3 candidates) can be combined with every local search tier implementation (4 candidates) to form 12 distinct solvers. Some implementations are not suited for combination due to duplication in functionality between the local search and population-based tiers and only 10 two-tiered solvers will be implemented.

Since the 2-opt neighbourhood definition is used in the particle swarm population-based tier implementation and the 2-opt inter- and extrapolation GA, combining the 2-opt neighbourhood search with either of these population-based tiers would result in duplication of functionality. Therefore these combinations are excluded from the project.

In addition to the two-tiered solvers developed for this project, two additional solvers will be implemented for benchmark purposes. These solvers are an implementation of the simulated annealing heuristic for the sequential ordering problem and a GA implementation using a technique called edge assembly crossover (EAX) developed by Nagata & Kobayashi (1997). Both these solvers are known to be effective at solving sequencing problems and will be used as a benchmark against which to compare the other solvers.

Furthermore, each tier will be independently evaluated for comparison yielding a further 7 solvers, two benchmark methods will be implemented and a random solution guessing algorithm will be implemented to establish a minimum performance level required from any other solver. A total of 20 solvers will be implemented.

The complexity of a two-tiered solver is dominated by the tier with the highest computational complexity. Since all the population-based tier implementations have a time complexity of $O(l^2)$, and the local search tier implementations are all either $O(l)$, $O(l^2)$ or $O(l^3)$, the time complexity of the two-tiered solvers will either be $O(l^2)$ or $O(l^3)$, depending on the local search tier implementation.

The solver combinations are shown in Table 5.

Table 5: Solver Implementations for Comparison

Solver Number	Population-based Tier	Local Search Tier	Complexity
Population-based tier implementations			
0	Repeated random guessing	None	$O(l)$
1	Particle swarm optimisation	None	$O(l^2)$
2	2-opt inter-and extrapolation GA	None	$O(l^2)$
3	Partially mapped crossover GA	None	$O(l^2)$
Local search tier implementations			
A	Repeated random guessing	2-opt neighbourhood search	$O(l)$
B	Repeated random guessing	Substring exchange neighbourhood search	$O(l^3)$
C	Repeated random guessing	Pair-wise exchange neighbourhood search	$O(l^2)$
D	Repeated random guessing	Single node displacement neighbourhood search	$O(l^3)$
Two-tiered combinations			
3A	Partially mapped crossover GA	2-opt neighbourhood search	$O(l^2)$
1B	Particle swarm optimisation	Substring exchange neighbourhood search	$O(l^3)$
2B	2-opt inter-and extrapolation GA	Substring exchange neighbourhood search	$O(l^3)$
3B	Partially mapped crossover GA	Substring exchange neighbourhood search	$O(l^3)$
1C	Particle swarm optimisation	Pair-wise exchange neighbourhood search	$O(l^2)$
2C	2-opt inter-and extrapolation GA	Pair-wise exchange neighbourhood search	$O(l^2)$
3C	Partially mapped crossover GA	Pair-wise exchange neighbourhood search	$O(l^2)$
1D	Particle swarm optimisation	Single node displacement neighbourhood search	$O(l^3)$
2D	2-opt inter-and extrapolation GA	Single node displacement neighbourhood search	$O(l^3)$
3D	Partially mapped crossover GA	Single node displacement neighbourhood search	$O(l^3)$
Benchmark solvers			
X	Edge assembly crossover GA	None	$O(l^3)$
Y	Repeated random guessing	Simulated Annealing	$O(l^2)$

4.5 COMPETING METHODS TO SOLVE THE SEQUENTIAL ORDERING PROBLEM

One of the objectives of the two-tiered solvers is that they should be competitive with other methods actually used in practice to solve sequencing problems. Two methods were selected as benchmarks against which to evaluate this objective. These methods are simulated annealing and a GA that uses a specialised crossover operation. The implementations of these methods are described in this section.

4.5.1 Simulated annealing for the sequential ordering problem

The simulated annealing algorithm follows the guidelines in Subsection 2.6.3b.) on page 44. The method uses a temperature parameter T to control the acceptance of new solutions during the search. Through a predetermined annealing schedule, T is reduced during the search, and as T becomes smaller, the probability of accepting a poor solution is reduced.

The pair-wise exchange neighbourhood described in Subsection 4.3.3 on page 88 is used as the neighbourhood relation.

Since the standard simulated annealing algorithm compares only objective function values when calculating the probability of accepting a new solution, some way is required to recognise improvements in the feasibility of solutions. The following method is used to include the level of infeasibility in the evaluation:

The degree of infeasibility is multiplied by the greatest transition cost in the sequential ordering problem's cost matrix and applied as a penalty to the objective function. This ensures that the simulated annealing algorithm will recognise a reduction in the degree of infeasibility as a significant improvement to a candidate solution.

Unlike the steepest descent methods described in Section 4.3 on page 85, simulated annealing does not evaluate all available neighbours, but randomly selects neighbours of the current solution until a suitable candidate is found. If the search is close to an optimum and the temperature parameter T is low, the probability of generating acceptable new solutions becomes small. To ensure that no computational effort is wasted in repeating neighbours, a method is required to cycle through all the neighbours without repetition, but also without a predictable pattern. This is done in the following way:

1. Randomly generate a sequence of numbers from 1 to l, where l is the sequence length required for a solution to the sequential ordering problem. This sequence is called the position sequence.

2. Randomly generate another sequence of numbers from 1 to *l*-1. This sequence is called the offset sequence.
3. The first pair of positions to be exchanged in the current candidate solution is determined as follows:
   a) The first position is equal to the first number of the position sequence.
   b) The second number is the sum of the first number of the position sequence and the first number of the offset sequence modulus *l*.
4. Each successive pair of positions is generated by using the next pair of numbers in the position and offset sequences.
5. When the end of the offset sequence is reached, the sequence is repeated.

This method will cycle through every pair of positions without repeating.

When *T*, the temperature parameter, reaches zero, this implementation will change to the steepest ascent method described in Section 4.3 on page 85 and it will continue until a local optimum is found.

The following program is used to implement the simulated annealing algorithm:

Pseudo-code for a simulated annealing solver for the sequential ordering problem

```
currentsolution ← candidatesolution
repeat
   T ← annealingSchedule(T)
   degreeofviolation ← degreeOfViolation(currentsolution)
   objectivefunctionvalue ← objectiveFunctionValue(currentsolution)
   score ← objectivefunctionvalue + objectivefunctionvalue*maxtransitioncost
   repeat
      generate a neighbour (use pair generation routing described above)
      neighbourdegreeofviolation ← degreeOfViolation(neighbour)
      neighbourobjectivefunctionvalue ← objectiveFunctionValue(neighbour)
      neighbourscore ← neighbourobjectivefunctionvalue +
                       objectivefunctionvalue*maxtransitioncost
      selectionprobability = 1 / (1 + e^((neighbourscore − score)/T))
      with a probability of selectionprobability: currentsolution ← neighbour
   until currentsolution = neighbour
until T = 0
optimise currentsolution using pair-wise node exchange neighbourhood search
return currentsolution
```

From a time complexity point of view, the only operations in the simulated annealing algorithm that are dependant on the length of the required solution sequence, are the pair generation routines described above and the evaluations of the objective function. Both these operations are $O(l)$. However, since this method ends by performing a pair-wise node exchange neighbourhood search, which is $O(l^2)$, this implementation is $O(l^2)$.

4.5.2 GA using edge assembly crossover (EAX) for the sequential ordering problem

Nagata and Kobayashi (1997) developed edge assembly crossover (EAX) as a method to generate tours for the travelling salesman problem. In this subsection, the crossover procedure will be introduced and the specific implementation of a GA using EAX to solve the sequential ordering problem will be described.

EAX generates "child" tours by first combining the edges of two "parent" tours to form a set of smaller disconnected cycles. These cycles are then combined to form a new tour. In the sequential ordering problem, solutions can be represented as Hamiltonian cycles on a directed graph by combining the start and end elements of the problem into a single node. A Hamiltonian cycle is a set of edges that form a closed path that passes through every node on the graph exactly once. Any sequence that is a valid solution to the sequential ordering problem can be expressed as a cycle that contains the combined start/end element.

EAX uses a construct called an AB-cycle to achieve its objective. An AB-cycle is a set of transitions or edges that are constructed as follows:

An AB-tour starts with an arbitrary edge from the one cycle and edges are then alternately selected from the two cycles so that each new edge shares the same destination or origin as the previous edge. It can be shown that each edge in the pair of tours is uniquely associated with an AB-cycle.

Any one of the original parents can be modified by applying an AB-cycle to it. An AB-cycle is applied to a graph by removing all corresponding edges from the graph and adding all non-corresponding edges to the graph. If the graph is a Hamiltonian cycle, the result is always a collection of one or more disconnected cycles that cover all the nodes of the graph. If an AB-cycle consists of two identical edges, any graph it is applied to, remains unchanged. Such an AB-cycle is called an "ineffective" AB-cycle. An AB-cycle that is not ineffective is called an "effective" AB-cycle.

Figure 14 shows how two parent cycles are broken down into two effective and three ineffective AB-cycles.

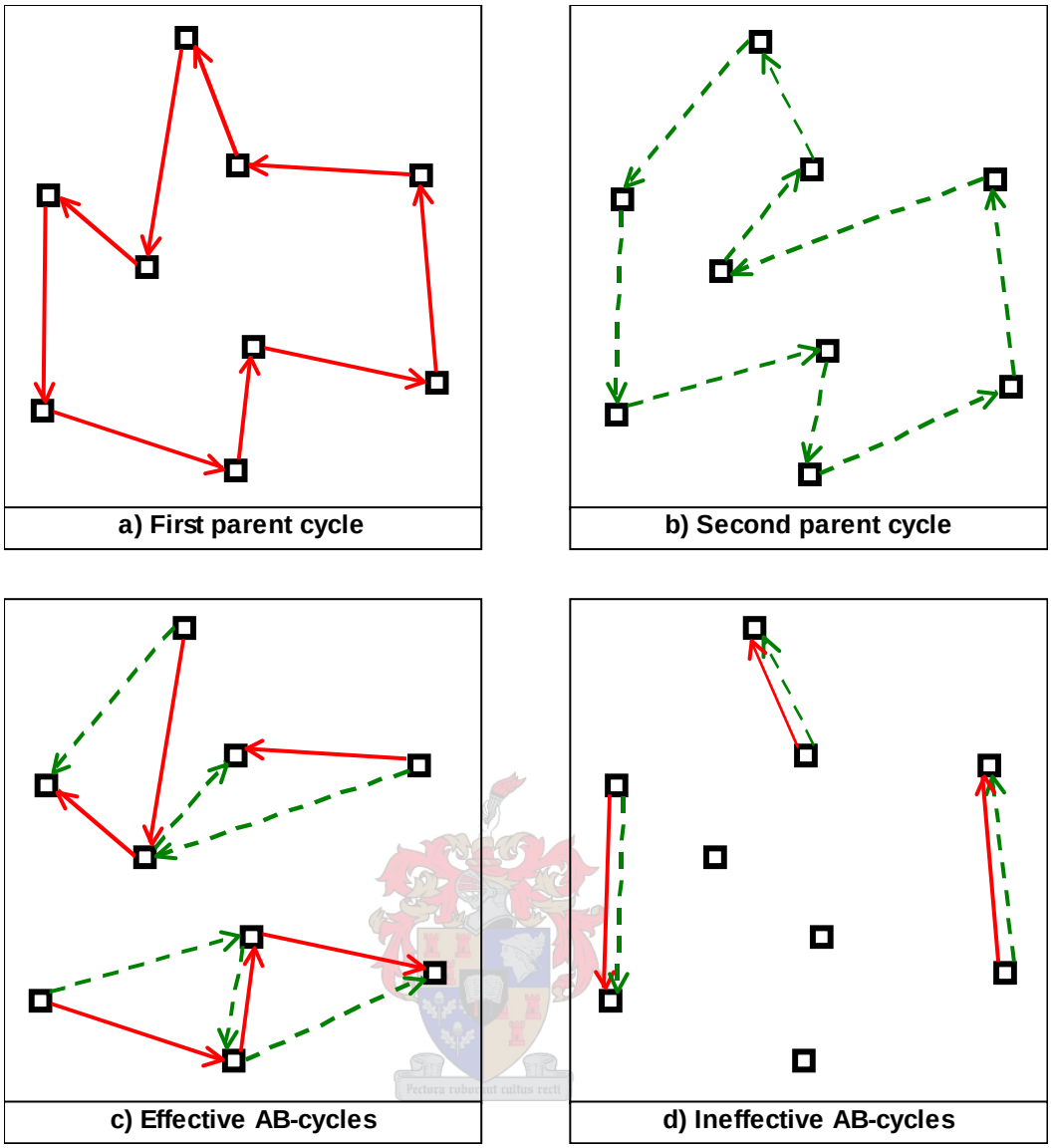


Figure 14: An Example of AB-cycle Generation

Figure 15 shows how an AB-cycle is applied to a cycle. In this example the result is a new cycle. In other cases it could result in a set of disjoint smaller cycles that cover the graph.

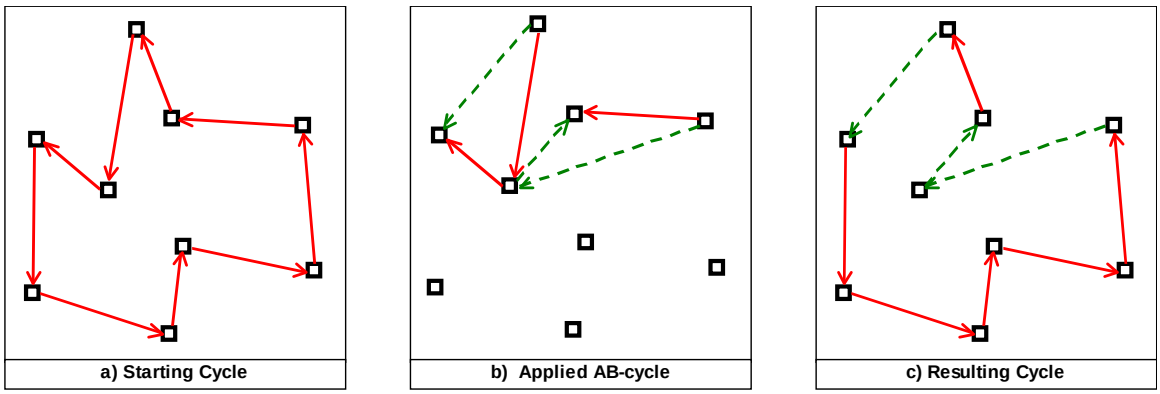


Figure 15: An Example of the Application of an AB-cycle to a Cycle

Once the effective AB-cycles are generated from the two parent cycles, the crossover method constructs a so-called E-set by selecting a number of effective AB-cycles. In the implementation used in this project, AB-cycles for the E-set are selected at random. All AB-cycles from the E-set are then applied to one of the parents to form a so-called "intermediate individual". An intermediate individual is a directed graph that contains a number of disconnected cycles that cover the node-set of the graph.

A cost function associated with intermediate individuals is calculated in the following way:

First the total transition cost of all the edges on the graph is determined. Penalties are then calculated for the cycle containing the starting node. This cycle is called the "initiating cycle". A penalty is calculated for each constraint where both the constrained nodes are on the initiating cycle. The penalty is calculated by multiplying the degree of violation of each constraint by the greatest transition cost in the transition cost matrix of the sequential ordering problem. The cost function value is equal to the sum of the transition costs and all penalties of the initiating cycle.

Finally, the intermediate individual is converted into a Hamiltonian cycle by connecting all the individual cycles in the graph using the following method:

Starting with the initiating cycle, this cycle is connected to one of the other cycles by removing one edge from each of the cycles and reconnecting the resulting paths to form a new combined cycle. The combination is determined through an exhaustive search to identify the combination with the lowest cost. This combination procedure is repeated with the resulting intermediate individual until only one cycle remains. This cycle is the result of the crossover procedure.

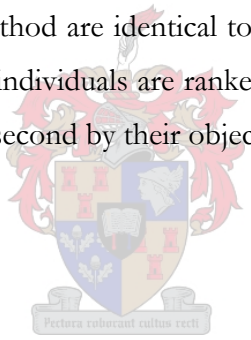
This GA implementation is similar to both the GA-based population-based tier implementations described in Section 4.2 on page 72, and uses three stages to update its population of candidate solutions:

1. **Generate offspring**: Using the EAX crossover procedure, the GA produces candidate offspring. Each sequence in the population has a probability of p_c to be selected for crossover and parents are paired off at random. Parents remain candidates and are not discarded during this stage.

2. **Introduce mutation**: Each element in each sequence in the resulting population can be selected for mutation with a probability of p_m. When an element is selected for mutation, it is randomly relocated within its sequence.

3. **Selection**: Candidates are selected from the new population using rank-space selection (see Subsection 2.4.3b.) on page 27) until the next generation reaches a predetermined population size. Each candidate can only appear once in the new generation.

The selection criteria using this method are identical to those used in the GA-based population-based tier implementations: When individuals are ranked by fitness, they are first ranked by their degree of constraint violation, and second by their objective function values.



The following pseudo-code describes the program-flow of this implementation:

Pseudo-code for the GA-based solver using edge assembly crossover

```
 1: generate a random initial population
 2: do
 3:    parentset ← empty
 4:    for each candidate in population do
 5:       add candidate to parentset with probability p_c
 6:    next candidate
 7:    repeat
 8:       randomly select parent1 and parent2 from parentset
 9:       offspring ← edgeAssemblyCrossover(parent1,parent2)
10:       add offspring to population
11:       remove parent1 and parent2 from parentset
12:    until number of elements in parentset ≤ 0
13:    for each candidate in population do
14:       for each element in candidate do
15:          with probability p_m, randomly relocate element in candidate
16:       next element
17:    next candidate
18:    newpopulation ← empty
19:    repeat
20:       selectedCandidate ← rankSpaceSelection(population)
21:       add selectedCandidate to newpopulation
22:       remove selectedCandidate from population
23:    until the size of newpopulation = a predetermined popsize
24:    population ← newpopulation
25: until maximum iterations or minimum error criteria is met
26: return best candidate in population
```

The complexity of this implementation is dominated by the crossover operator, which is $O(l^3)$, therefore this implementation has a complexity of $O(l^3)$ per generation.

4.6 CHAPTER SUMMARY

This chapter discussed the implementation of several heuristics for solving the sequential ordering problems. Central to this chapter was the development of a series of two-tiered solvers.

The first section contains an overview of some key strategies that were used to implement the solvers: First the data structures used to represent the sequential ordering problem were determined. Second, the programmatic structuring of the population-based and local search tiers was formalised to ensure that different implementations of these tiers could interact predictably. Third, a strategy for dealing with precedence constraints was determined. This strategy is used in all the implementations of the two-tiered solvers. Fourth, diversity measures were identified to measure and direct changes to the population of candidate solutions maintained by population-based tiers. Finally, a method was proposed to measure the performance of individual tiers independently of a two-tiered solver that they would otherwise be part of.

The second section specified the detailed implementation of three population-based tiers: The first implementation was an implementation of the particle swarm optimiser that uses a series of 2-opt transitions to "move" particles from one position to the other. The second implementation is a genetic algorithm that uses a 2-opt inter- extrapolation mechanism to generate new candidates. The third and final implementation was a genetic algorithm that uses the partially mapped crossover operator. Both the genetic algorithm implementations use rank-space selection to promote population diversity and prevent convergence to a single solution.

In Section 4.3., four local search heuristics were identified for use in the two-tiered optimisers. All these heuristics were deterministic and all follow the steepest-descent search strategy. The first local search heuristic was the 2-opt local search heuristic. The second heuristic used substring inversion to generate neighbours. The third heuristic used pair-wise node exchange to generate neighbours and the final heuristic generated new neighbours through the displacement of a single element in a sequence.

In Section 4.4., the two-tiered implementations were identified that can be constructed using the population-based and local search tiers developed earlier in this chapter.

Section 4.5. introduced two competing heuristics. The first heuristic is a form of simulated annealing that is related to the pair-wise node exchange local search algorithms. The second heuristic is a genetic algorithm that uses a specialised crossover operator called edge assembly crossover.

In the next chapter the performance of all the solvers developed in this chapter is analysed.

5 RESULTS

In this chapter, the performance of solvers that were constructed in Chapter 4 is analysed.

In the first section the experimental procedure is described. The problems that were used as benchmarks are identified, the population sizes for the two-tiered solvers are chosen and the termination criteria for the trials are specified.

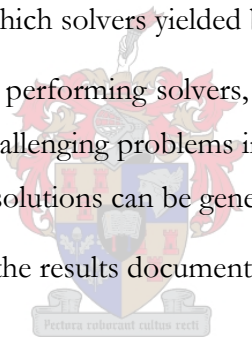
In Section 5.2. the computational requirements of each solver are evaluated in terms of a function of the problem size.

In Section 5.3. the ability of solvers to identify global optima is evaluated. By analysing the experimental data generated through the procedure specified in the first section, the time required by various solvers to reach global optima is compared. Solvers are then presented in order of performance for every benchmark problem for which optima were found.

In Section 5.4. the quality of solutions generated by each solver over a limited time period is compared. This analysis indicates which solvers yielded better results over a limited time period.

Finally in Section 5.5. the two best performing solvers, along with the two benchmark methods, are executed on two of the most challenging problems in the benchmark set for an extended time period of 5 hours to see if optimal solutions can be generated for these problems.

Section 5.6 contains a summary of the results documented in this chapter.



5.1 EXPERIMENTAL PARAMETERS

This section describes the experimental conditions under which the solvers were evaluated. The benchmark problems on which the solvers are tested are identified and their properties are stated. The parameters of the solvers will be specified, and finally, the number of trials and termination criteria will be specified.

5.1.1 Benchmark problems

Ten benchmark problems were selected from TSPLIB (see Subsection 3.4.1 on page 62). The problems were selected to cover a range of sizes from 14 to 100 nodes and to include all the problems with symmetric transition costs in this range.

For all except two of the selected problems, the optimum is known. For the two exceptions, upper and lower bounds on the optimal objective function value are known. For the termination and evaluation criteria, the upper bound is treated as if it were the optimal objective function value.

The selected benchmark problems from TSPLIB and their properties are shown in Table 6 below:

			Optimal Value			
Name	**Number of Locations**	**Number of Constraints**	**Lower Bound**	**Upper Bound**	**Exact**	**Symmetrical**
ESC12	14	7			1,675	Yes
ESC25	27	9			1,681	
br17.12	17	12			55	
ft53.1	54	12	7,438	7,570		
ft70.1	71	17			39,313	
p43.1	44	9			27,990	
prob.42	42	10			243	Yes
prob.100	100	41	1,024	1,385		Yes
rbg048a	50	192			351	
rbg050c	52	256			467	

Table 6: Instances of the Sequential Ordering Problem Selected as Benchmarks

5.1.2 Solver Parameters

Some of the solvers developed in Chapter 4 require parameters that guide their behaviour. The parameters selected for the experiment are the following:

All population-based tiers and solvers have a population size of 50. This number was arbitrarily chosen to eliminate the size parameter as a possible source of differentiation between solver performances.

For all genetic algorithm based tiers, the probability of mutation, p_m, was fixed at 0.01 and the probability, p_c, of crossover was fixed at 0.5.

For the rank-space selection method used in the genetic algorithms, the probability of selection is exponentially scaled with a 0.5 probability of selection for the top ranked candidate, a 0.25 probability of selection for the second and 0.5^n for the nth ranked candidate.

The simulated annealing algorithm's annealing schedule is set to run for 30 seconds per iteration, it starts at a temperature of 10 times the maximum transition cost and progresses down to 0.1 on an exponential cooling curve.

Parameters were arbitrarily selected and are not optimal.

5.1.3 Trials and Termination Criteria

Every solver is allowed ten trial runs on every problem.

The termination criteria for a trial are (a) more that 300 seconds have passed from the start of the first iteration to the completion of the last iteration, or (b) a global optimum has been identified.

For this experiment, the result is recorded of every iteration at which an improvement was found. The final iteration in each trail is also recorded.

The following information is recorded:

- The problem on which the solver is running.
- The solver.
- The number of the trial.
- The time it took from the start to the end of the trial.
- The number of the iteration in the trial.
- The degree of constraint violation of the current best solution.
- The objective function value of the best current solution.
- The best current solution.
- The current status of the trial: Running, Optimum Found or No Optimum Found.

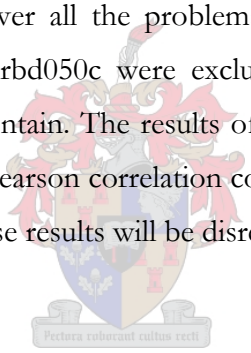
A full set of data has been generated and the results are included in soft copy.

5.2 COMPUTATIONAL REQUIREMENTS

The number of iterations that solvers achieve during each trial gives an abstract indication of the time required to execute an iteration on a given problem. In this section, the time required by a solver to complete a single iteration of a problem of a given size, is evaluated.

For each solver, the average time required to execute an iteration on a given problem is estimated by dividing the time required to complete the trial by the number of iterations completed during the trial. Since ten trials were executed for each problem-solver combination, an average time per iteration can be determined by dividing the total time spent on the ten trials by the total number of iterations that were completed.

It is assumed that the time required to execute a single iteration can be modelled by the following equation $t = m \cdot n^k$, where t is the time required to complete an iteration, n is the number of locations represented in the problem and m and k are constants. This equation can also be expressed as $\ln(t) = k \cdot \ln(n) + \ln(m)$. Since $\ln(t)$ and $\ln(n)$ are known, k and $\ln(m)$ can be determined by linear regression over all the problems and for each solver. For this exercise problems prob.100, rbg048a and rbd050c were excluded due to the disproportionately high number of constraints that they contain. The results of the regression are shown in Table 7. In instances where the square of the Pearson correlation coefficient, R^2, < 0.95 the regression model is considered unacceptable and those results will be disregarded.



Solver	m (in seconds)	k	R^2
0	263.3 $x10^9$	1.092	0.968
1	897.0 $x10^9$	4.413	0.998
2	63.81 $x10^9$	4.423	0.999
3	483.8 $x10^6$	0.679	0.987
A	29.18 $x10^9$	3.479	0.969
B	46.02 $x10^9$	4.045	0.987
C	12.49 $x10^9$	4.176	0.989
D	25.14 $x10^9$	4.222	0.987
3A	2.564 $x10^6$	2.933	0.961
1B	1.487 $x10^6$	4.410	0.996
2B	5.087 $x10^6$	3.732	0.968
3B	9.879 $x10^6$	3.446	0.949
1C	710.4 $x10^9$	4.544	0.998
2C	1.197 $x10^6$	3.942	0.976
3C	4.472 $x10^6$	3.425	0.940
1D	1.111 $x10^6$	4.495	0.998
2D	4.392 $x10^6$	3.731	0.941
3D	23.78 $x10^6$	3.104	0.882
X	3.897$x10^6$	2.479	0.968
Y	29.40	6.731$x10^{-3}$	0.421

Table 7: Linear Regression of the log of Iteration Times over the logarithm of the Problem Size

From the results of the linear regression, one may assume that the model is valid for the majority of the solvers. The most efficient of the solvers in terms of scalability is solver 3, for which the time required to complete an iteration increased sub-linearly to the size of the problem. The time complexity of solver Y is known to be almost independent of the size of the problems since the annealing schedule is limited to 30 seconds.

The time required to complete an iteration for all other solvers for which this model is valid, can be assumed to be a polynomial function of n and at most $O(n^5)$.

5.3 IDENTIFICATION OF GLOBAL OPTIMA

In this subsection, the ability of solvers to find global optima is compared. In this experiment, the solvers only managed to identify global optima in three of the ten benchmark problems. They were: br17.12, ESC12 and ESC25, the three easiest problems.

The Wilcoxon rank-sum test (Walpole & Myers 1990, pp. 626-630), a non-parametric statistical method, was used to compare the results and identify significant differences in the median of the times required to find the global optima. The test was conducted at a 0.05 level of significance.

Non-parametric tests are better suited to testing hypotheses than parametric tests, like the *t*-test, when fewer than 30 observations are available and there is no reason to believe that the data is distributed normally, as is the case in this experiment.

The results of these tests are shown in a "dominance matrix". The matrix shows empty blocks where the results from two solvers are not significantly different. If the median time required by the first solver is significantly less than the median time required by the second solver, the result is indicated by a minus sign. If the median time required by the first solver is significantly more that the median time required by the second solver, the result is indicated by a plus sign. The rows and columns of solvers that failed to return any results in the given time period are marked in grey.

The solvers are ranked according to their performance.

Finally, a synergy matrix shows whether or not the two-tiered solvers manage to outperform their component tiers. A tick (✓) on the intersection of the components on the synergy matrix indicates that a two-tiered solver managed to outperform both its components, while a cross (✗) indicates that both the components managed to outperform the solver. No symbol indicates that the solver did not manage to significantly outperform both its component tiers, but both the component tiers could not outperform the solver either.

The results of the analyses are shown in Table 8, Table 9 and Table 10.

Solver Dominance Matrix for Problem : br17.12

First Solver \ Second Solver	0	1	2	3	A	B	C	D	3A	1B	2B	3B	1C	2C	3C	1D	2D	3D	X	Y
Y	-	-	-	-	+	+	+	+	+	+	+	+	+	+	+	+	+	+		
X	-	-	-	-	+	+	+	+	+	+	+	+	+	+	+	+	+	+		
3D	-	-	-	-	-	-	-	-	+	+	+	+	+	+	+	+	+		-	-
2D	-	-	-	-	-	-	-	-	-	+	+	-	+	+	-	+		-	-	-
1D	-	-	-	-	-	-	-	-	-	+	-	-	+	-	-		-	-	-	-
3C	-	-	-	-	-	-	-	-	+	+	+	+	+	+		+	+	-	-	-
2C	-	-	-	-	-	-	-	-	-	+	+	-	+		-	+	-	-	-	-
1C	-	-	-	-	-	-	-	-	-	+	-	-		-	-	-	-	-	-	-
3B	-	-	-	-	-	-	-	-	+	+	+		+	+	-	+	+	-	-	-
2B	-	-	-	-	-	-	-	-	-	+		-	+	-	-	+	-	-	-	-
1B	-	-	-	-	-	-	-	-	-		-	-	-	-	-	-	-	-	-	-
3A	-	-	-	-	-	-	-	-		+	+	-	+	+	-	+	+	-	-	-
D	-	-	-	-	+	+	+		+	+	+	+	+	+	+	+	+	+	-	-
C	-	-	-	-	+	+		-	+	+	+	+	+	+	+	+	+	+	-	-
B	-	-	-	-	+		-	-	+	+	+	+	+	+	+	+	+	+	-	-
A	-	-	-	-		-	-	-	+	+	+	+	+	+	+	+	+	+	-	-
3					+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
2					+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
1					+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
0					+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+

Solver Performance Rank

Solver	Rank	Best Time
1B	1	0.469
1C	2	0.296
1D	3	0.422
2B	4	0.391
2C	5	0.156
2D	6	0.328
3A	7	0.187
3B	8	0.375
3C	9	0.156
3D	10	0.328
A	11	0.859
B	12	0.016
C	13	0.031
D	14	0.016
X	15	53.281
Y	15	30.015
0	17	Not Found
1	17	Not Found
2	17	Not Found
3	17	Not Found

Synergy Matrix for Problem : br17.12

Population-based Tier \ Local Search Tier	A	B	C	D
3	✓	✓	✓	✓
2		✓	✓	✓
1		✓	✓	✓

Table 8: Performance Analysis - Identification of Global Optima on br17.12

Solver Dominance Matrix for Problem : ESC12

First Solver \ Second Solver	0	1	2	3	A	B	C	D	3A	1B	2B	3B	1C	2C	3C	1D	2D	3D	X	Y
Y					+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	
X	-	-	-	-	+	+	+	+	+	+	+	+	+	+	+	+	+	+		-
3D	-	-	-	-	-	-	-	-	+	+	+	+	+	+	+	+	+		-	-
2D	-	-	-	-	-	-	-	-	-	+	+	-	+	+	-	+		-	-	-
1D	-	-	-	-	-	-	-	-	-	+	-	-	+	-	-		-	-	-	-
3C	-	-	-	-	-	-	-	-	+	+	+	+	+	+		+	+	-	-	-
2C	-	-	-	-	-	-	-	-	-	+	+	-	+		-	+	-	-	-	-
1C	-	-	-	-	-	-	-	-	-	+	-	-		-	-	-	-	-	-	-
3B	-	-	-	-	-	-	-	-	+	+	+		+	+	-	+	+	-	-	-
2B	-	-	-	-	-	-	-	-	-	+		-	+	-	-	+	-	-	-	-
1B	-	-	-	-	-	-	-	-	-		-	-	-	-	-	-	-	-	-	-
3A	-	-	-	-	-	-	-	-		+	+	-	+	+	-	+	+	-	-	-
D	-	-	-	-	+	+	+		+	+	+	+	+	+	+	+	+	+	-	-
C	-	-	-	-	+	+		-	+	+	+	+	+	+	+	+	+	+	-	-
B	-	-	-	-	+		-	-	+	+	+	+	+	+	+	+	+	+	-	-
A	-	-	-	-		-	-	-	+	+	+	+	+	+	+	+	+	+	-	-
3					+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	
2					+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	
1					+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	
0					+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	

Solver Performance Rank

Solver	Rank	Best Time
1B	1	0.125
1C	2	0.078
1D	3	0.125
2B	4	0.094
2C	5	0.046
2D	6	0.094
3A	7	0.016
3B	8	0.093
3C	9	0.047
3D	10	0.093
A	11	0.125
B	12	0.015
C	13	0.015
D	14	0.015
X	15	13.703
0	16	7.859
1	16	Not Found
2	16	Not Found
3	16	65.094
Y	16	30.016

Synergy Matrix for Problem : ESC12

Population-based Tier \ Local Search Tier	A	B	C	D
3	✓	✓	✓	✓
2		✓	✓	✓
1		✓	✓	✓

Table 9: Performance Analysis - Identification of Global Optima on ESC12

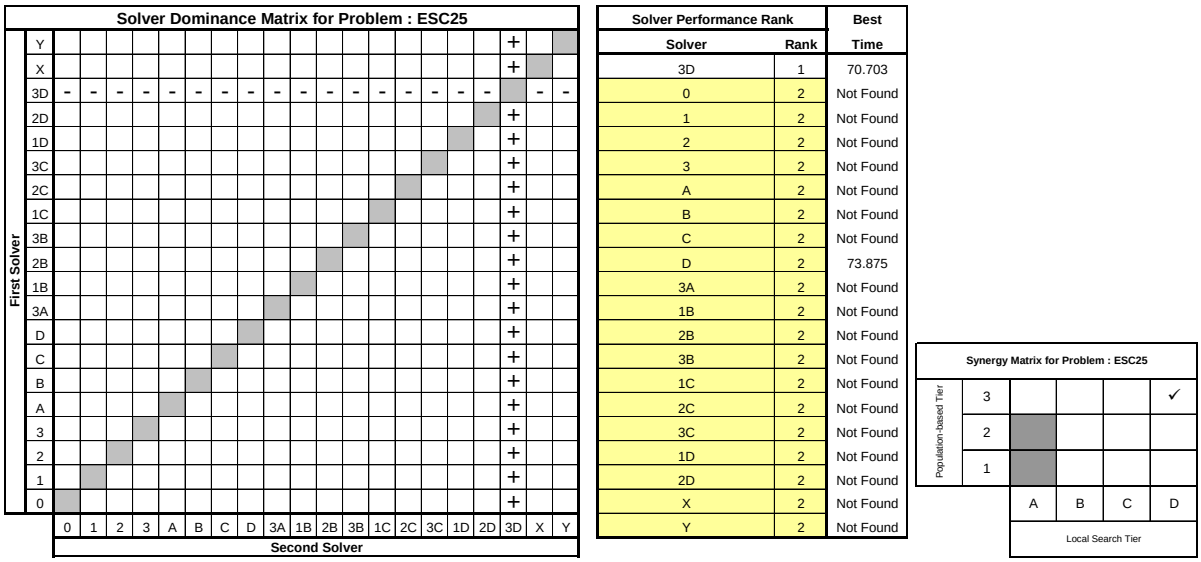


Table 10: Performance Analysis - Identification of Global Optima on ESC25

For problems br17.12 and ESC12, the ranks of the solvers are similar, with 1B being the highest ranked in both cases. For these problems, all the two-tiered solvers outperformed their component tiers.

In problem ESC25, only solver 3D could consistently identify the global optima. Therefore, this is also the only solver that outperformed its components.

5.4 SOLVER PERFORMANCE IN LIMITED TIME TRIALS

In this section, the quality of various solutions achieved in limited time trials is compared. For each solver and each problem, the medians of the objective function values achieved within a 300 second trial, are compared to determine significant differences in solver performance.

The same method used in the previous section, the Wilcoxon rank-sum test (Walpole & Myers 1990, pp. 626-630), was used to compare the results and identify significant differences in the median of objective function values. The test was conducted at a 0.05 level of significance.

The results of the limited time trial are analysed using the same tools that were used to analyse times required to reach global optima in the previous section. The results are shown in a "dominance matrix". The matrix shows empty blocks where the results from two solvers are not significantly different. If the median objective function value achieved by the first solver is significantly less than the median objective function value achieved by the second solver, the result is indicated by a minus sign and when the median objective function value achieved by the first solver is significantly more than the median objective function value achieved by the second solver, the result is indicated by a plus sign.

Solvers are ranked according to their performance.

Finally, a synergy matrix shows whether or not the two-tiered solvers manage to outperform their component tiers. A tick (✓) on the intersection of the components on the synergy matrix indicates that a two-tiered solver managed to outperform both its components, while a cross (✗) indicates that both the components managed to outperform the solver. No symbol indicates that the solver did not manage to significantly outperform both its component tiers, but both the component tiers could not outperform the solver either.

The results of the analyses are shown in Tables 11 through 20.

Solver Dominance Matrix for Problem : br17.12

First Solver \ Second Solver	0	1	2	3	A	B	C	D	3A	1B	2B	3B	1C	2C	3C	1D	2D	3D	X	Y
Y	-	-	-	-																
X	-	-	-	-																
3D	-	-	-	-																
2D	-	-	-	-																
1D	-	-	-	-																
3C	-	-	-	-																
2C	-	-	-	-																
1C	-	-	-	-																
3B	-	-	-	-																
2B	-	-	-	-																
1B	-	-	-	-																
3A	-	-	-	-																
D	-	-	-	-																
C	-	-	-	-																
B	-	-	-	-																
A	-	-	-	-																
3		-	-		+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
2	+	-		+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
1	+		+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
0		-	-		+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+

Solver Performance Rank

Solver	Rank	Best Score
A	1	55
B	1	55
C	1	55
D	1	55
3A	1	55
1B	1	55
2B	1	55
3B	1	55
1C	1	55
2C	1	55
3C	1	55
1D	1	55
2D	1	55
3D	1	55
X	1	55
Y	1	55
0	17	58
3	17	61
2	19	63
1	20	99
Optimum		55

Synergy Matrix for Problem : br17.12

Population-based Tier \ Local Search Tier	A	B	C	D
3				
2				
1				

Table 11: Performance Analysis – Limited Time Trial on br17.12

Solver Dominance Matrix for Problem : ESC12

First Solver \ Second Solver	0	1	2	3	A	B	C	D	3A	1B	2B	3B	1C	2C	3C	1D	2D	3D	X	Y
Y		-	-		+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	
X	-	-	-	-																-
3D	-	-	-	-																-
2D	-	-	-	-																-
1D	-	-	-	-																-
3C	-	-	-	-																-
2C	-	-	-	-																-
1C	-	-	-	-																-
3B	-	-	-	-																-
2B	-	-	-	-																-
1B	-	-	-	-																-
3A	-	-	-	-																-
D	-	-	-	-																-
C	-	-	-	-																-
B	-	-	-	-																-
A	-	-	-	-																-
3		-	-		+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
2	+	-		+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
1	+		+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
0		-	-		+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+

Solver Performance Rank

Solver	Rank	Best Score
A	1	1675
B	1	1675
C	1	1675
D	1	1675
3A	1	1675
1B	1	1675
2B	1	1675
3B	1	1675
1C	1	1675
2C	1	1675
3C	1	1675
1D	1	1675
2D	1	1675
3D	1	1675
X	1	1675
0	16	1675
3	16	1675
Y	16	1675
2	19	1704
1	20	1781
Optimum		1675

Synergy Matrix for Problem : ESC12

Population-based Tier \ Local Search Tier	A	B	C	D
3				
2				
1				

Table 12: Performance Analysis – Limited Time Trial on ESC12

Solver Dominance Matrix for Problem : ESC25

First Solver \ Second Solver	0	1	2	3	A	B	C	D	3A	1B	2B	3B	1C	2C	3C	1D	2D	3D	X	Y
Y	-	-	-	-	-	-	+	+		-			+	+	+	+	+	+	-	
X	-	-	-	-	+	+	+	+	+	+	+	+	+	+	+	+	+	+		+
3D	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-		-	-
2D	-	-	-	-	-	-	-		-	-	-	-	-	-	-	-		+	-	-
1D	-	-	-	-	-	-	-	+	-	-	-	-	-	-			+	+	-	-
3C	-	-	-	-	-	-	-	+	-	-	-	-	-	-			+	+	-	-
2C	-	-	-	-	-	-		+	-	-	-	-	-		+	+	+	+	-	-
1C	-	-	-	-	-	-	+	+	-	-	-	-		+	+	+	+	+	-	-
3B	-	-	-	-	-	-	+	+	-	-			+	+	+	+	+	+	-	
2B	-	-	-	-	-	-	+	+	-	-			+	+	+	+	+	+	-	
1B	-	-	-	-		+	+	+	+		+	+	+	+	+	+	+	+	-	+
3A	-	-	-	-	-		+	+		-	+	+	+	+	+	+	+	+	-	
D	-	-	-	-	-	-	-		-	-	-	-	-	-	-	-		+	-	-
C	-	-	-	-	-	-		+	-	-	-	-	-		+	+	+	+	-	-
B	-	-	-	-	-		+	+		-	+	+	+	+	+	+	+	+	-	+
A	-	-	-	-		+	+	+	+		+	+	+	+	+	+	+	+	-	+
3		-	-		+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
2	+	-		+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
1	+		+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
0		-	-		+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+

Solver Performance Rank		Best
Solver	Rank	Score
3D	1	1681
D	2	1681
2D	2	1747
3C	4	1730
1D	4	1756
C	6	1980
2C	6	1756
1C	8	2211
2B	9	2647
3B	9	2784
Y	11	2580
B	12	3267
3A	12	3095
A	14	3486
1B	14	3363
X	16	4187
0	17	6145
3	17	6089
2	19	7096
1	20	9080
Optimum		1681

Synergy Matrix for Problem : ESC25

Population-based Tier \ Local Search Tier	A	B	C	D
3	✓	✓	✓	✓
2		✓		
1				

Table 13: Performance Analysis – Limited Time Trial on ESC25

Solver Dominance Matrix for Problem : ft53.1

First Solver \ Second Solver	0	1	2	3	A	B	C	D	3A	1B	2B	3B	1C	2C	3C	1D	2D	3D	X	Y
Y	-	-	-	-	-	-	+	+	-	-	-		+	+	+	+	+	-		
X	-	-	-	-	+	+	+	+	+	+	+	+	+	+	+	+	+		+	+
3D	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-			-	-
2D	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-			-	-
1D	-	-	-	-	-	-	-	+	-	-	-	-	-	-			+	+	-	-
3C	-	-	-	-	-	-	-	+	-	-	-	-	-	-			+	+	-	-
2C	-	-	-	-	-	-	-	+	-	-	-	-	-		+	+	+	+	-	-
1C	-	-	-	-	-	-	+	+	-	-	-	-		+	+	+	+	+	-	-
3B	-	-	-	-	-	-	+	+	-	-			+	+	+	+	+	+	-	
2B	-	-	-	-	-	-	+	+	-	-			+	+	+	+	+	+	-	+
1B	-	-	-	-	-		+	+	-		+	+	+	+	+	+	+	+	-	+
3A	-	-	-	-	-	+	+	+		+	+	+	+	+	+	+	+	+	-	+
D	-	-	-	-	-	-	-		-	-	-	-	-	-	-	-	+	+	-	-
C	-	-	-	-	-	-		+	-	-	-	-	-		+	+	+	+	-	-
B	-	-	-	-	-		+	+	-		+	+	+	+	+	+	+	+	-	+
A	-	-	-	-		+	+	+	+	+	+	+	+	+	+	+	+	+	-	+
3		-	-		+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
2	+	-		+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
1	+		+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
0		-	-		+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+

Solver Performance Rank		Best
Solver	Rank	Score
2D	1	7900
3D	1	7789
D	3	8118
3C	4	8322
1D	4	8254
2C	6	8529
C	7	8975
1C	8	9468
3B	9	9911
Y	9	9453
2B	11	10153
B	12	10939
1B	12	10670
3A	14	14506
A	15	16036
X	16	17759
0	17	20201
3	17	19596
2	19	23562
1	20	24073
Best Known Solution		7570

Synergy Matrix for Problem : ft53.1

Population-based Tier \ Local Search Tier	A	B	C	D
3	✓	✓	✓	✓
2		✓	✓	✓
1				

Table 14: Performance Analysis – Limited Time Trial on ft53.1

Solver Dominance Matrix for Problem : ft70.1

First Solver \ Second Solver	0	1	2	3	A	B	C	D	3A	1B	2B	3B	1C	2C	3C	1D	2D	3D	X	Y
Y	-	-	-	-	-	-	+	+	-	-	-	-	+	+	+	+	+	+	-	
X	-	-	-	-	+	+	+	+	+	+	+	+	+	+	+	+	+	+		+
3D	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-		-	-
2D	-	-	-	-	-	-	-		-	-	-	-	-	-	-	-		+	-	-
1D	-	-	-	-	-	-	-		-	-	-	-	-	-	-		+	+	-	-
3C	-	-	-	-	-	-	-		-	-	-	-	-	-			+	+	-	-
2C	-	-	-	-	-	-	-	+	-	-	-	-	-		+	+	+	+	-	-
1C	-	-	-	-	-	-	+	+	-	-	-	-		+	+	+	+	+	-	-
3B	-	-	-	-	-	-	+	+	-	-			+	+	+	+	+	+	-	+
2B	-	-	-	-	-	-	+	+	-	-			+	+	+	+	+	+	-	+
1B	-	-	-	-	-		+	+	-		+	+	+	+	+	+	+	+	-	+
3A	-	-	-	-	-	+	+	+		+	+	+	+	+	+	+	+	+	-	+
D	-	-	-	-	-	-	-		-	-	-	-	-	-	-			+	-	-
C	-	-	-	-	-	-		+	-	-	-	-	-	-	-			+	-	-
B	-	-	-	-	-		+	+	-		+	+	+	+	+	+	+	+	-	+
A	-	-	-	-		+	+	+	+	+	+	+	+	+	+	+	+	+	-	+
3	-	-	-		+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
2	+			+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
1	+			+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
0		-	-	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+

Solver Performance Rank		Best
Solver	Rank	Score
3D	1	40989
2D	2	41604
D	3	41301
3C	3	41918
1D	3	41928
2C	6	42185
C	7	43649
1C	8	43336
Y	9	44137
2B	10	49389
3B	10	49076
B	12	50968
1B	12	49908
3A	14	54712
A	15	57026
X	16	62321
3	17	65583
0	18	66165
1	19	69842
2	19	67855
Optimum		39313

Synergy Matrix for Problem : ft70.1

Population-based Tier \ Local Search Tier	A	B	C	D
3	✓	✓	✓	✓
2		✓	✓	
1				

Table 15: Performance Analysis – Limited Time Trial on ft70.1

Solver Dominance Matrix for Problem : p43.1

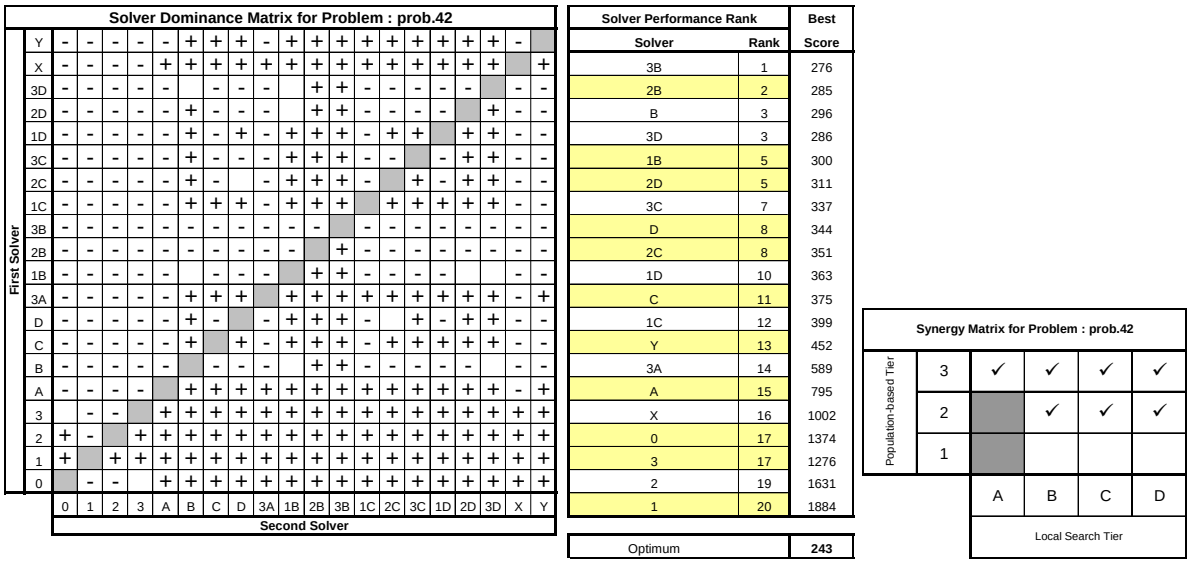
First Solver \ Second Solver	0	1	2	3	A	B	C	D	3A	1B	2B	3B	1C	2C	3C	1D	2D	3D	X	Y
Y	-	-	-	-	-	+	+	+	-	+	+	+	+	+	+	+	+	-		
X	-	-	-	-		+	+	+	+	+	+	+	+	+	+	+	+			+
3D	-	-	-	-	-		-	-	-	-			-	-		-	-		-	-
2D	-	-	-	-	-		-	-	-		+	+	-		+	-		+	-	-
1D	-	-	-	-	-	+			-	+	+	+	-	+	+		+	+	-	-
3C	-	-	-	-	-	-	-	-	-	-			-	-		-	-		-	-
2C	-	-	-	-	-		-	-	-		+	+	-		+	-		+	-	-
1C	-	-	-	-	-	+	+	+	-	+	+	+		+	+	+	+	+	-	-
3B	-	-	-	-	-	-	-	-	-	-			-	-		-	-		-	-
2B	-	-	-	-	-		-	-	-	-			-	-		-	-		-	-
1B	-	-	-	-	-		-	-	-		+	+	-		+	-		+	-	-
3A	-	-	-	-	-	+	+	+		+	+	+	+	+	+	+	+	+	-	+
D	-	-	-	-	-	+			-	+	+	+	-	+	+		+	+	-	-
C	-	-	-	-	-	+			-	+	+	+	-	+	+		+	+	-	-
B	-	-	-	-	-		-	-	-			+	-		+	-			-	-
A	-	-	-	-		+	+	+	+	+	+	+	+	+	+	+	+	+		+
3		-	-		+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
2	+	-		+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
1	+		+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
0		-	-		+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+

Solver Performance Rank		Best
Solver	Rank	Score
3B	1	28140
3C	1	28145
2B	3	28140
3D	3	28140
B	5	28170
1B	5	28180
2C	5	28195
2D	5	28175
C	9	28245
D	9	28260
1D	9	28240
1C	12	28295
Y	13	28435
3A	14	29200
A	15	29955
X	15	29715
0	17	31845
3	17	31790
2	19	58830
1	20	59990
Optimum		27990

Synergy Matrix for Problem : p43.1

Population-based Tier \ Local Search Tier	A	B	C	D
3	✓	✓	✓	✓
2			✓	✓
1				

Table 16: Performance Analysis – Limited Time Trial on p43.1



Solver Dominance Matrix for Problem : prob.42

First Solver \ Second Solver	0	1	2	3	A	B	C	D	3A	1B	2B	3B	1C	2C	3C	1D	2D	3D	X	Y
Y	-	-	-	-	-	+	+	+	-	+	+	+	+	+	+	+	+	-		
X	-	-	-	-	+	+	+	+	+	+	+	+	+	+	+	+	+			+
3D	-	-	-	-	-		-	-	-		+	+	-	-	-	-	-		-	-
2D	-	-	-	-	-	+	-	-	-		+	+	-	-	-	-		+	-	-
1D	-	-	-	-	-	+	-	+	-	+	+	+	-	+	+		+	+	-	-
3C	-	-	-	-	-	+	-	-	-	+	+	+	-	-		-	+	+	-	-
2C	-	-	-	-	-	+	-		-	+	+	+	-		+	-	+	+	-	-
1C	-	-	-	-	-	+	+	+	-	+	+	+		+	+	+	+	+	-	-
3B	-	-	-	-	-	-	-	-	-	-	-		-	-	-	-	-	-	-	-
2B	-	-	-	-	-	-	-	-	-	-		+	-	-	-	-	-	-	-	-
1B	-	-	-	-	-		-	-	-		+	+	-	-	-	-			-	-
3A	-	-	-	-	-	+	+	+		+	+	+	+	+	+	+	+	+	-	+
D	-	-	-	-	-	+	-		-	+	+	+	-		+	-	+	+	-	-
C	-	-	-	-	-	+		+	-	+	+	+	-	+	+	+	+	+	-	-
B	-	-	-	-	-		-	-	-		+	+	-	-	-	-	-		-	-
A	-	-	-	-		+	+	+	+	+	+	+	+	+	+	+	+	+	-	+
3		-	-		+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
2	+	-		+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
1	+		+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
0		-	-		+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+

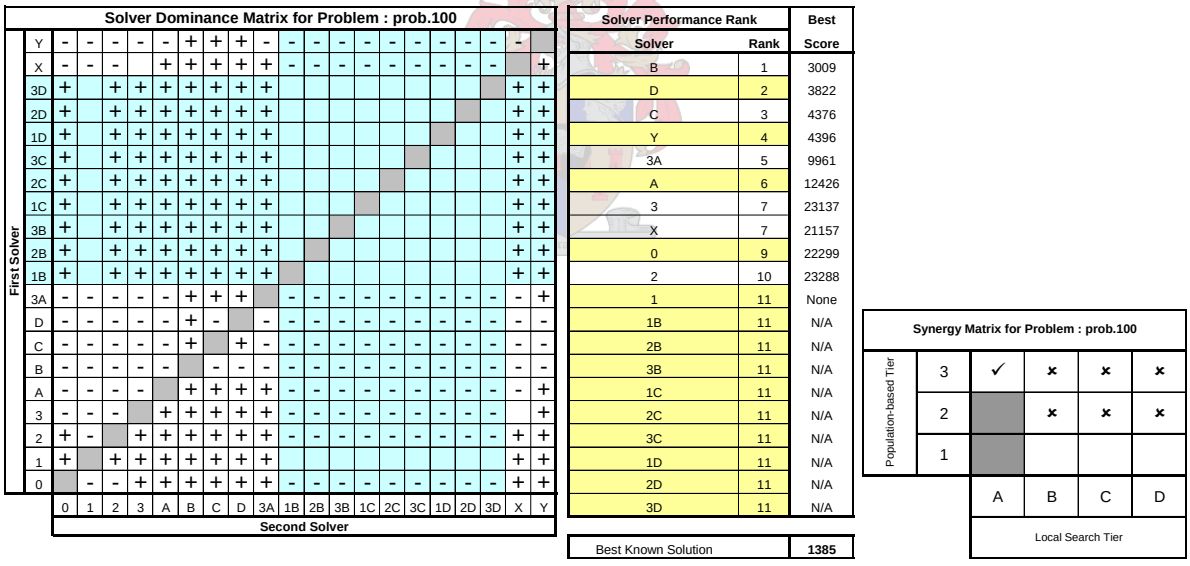
Solver Performance Rank		Best
Solver	Rank	Score
3B	1	276
2B	2	285
B	3	296
3D	3	286
1B	5	300
2D	5	311
3C	7	337
D	8	344
2C	8	351
1D	10	363
C	11	375
1C	12	399
Y	13	452
3A	14	589
A	15	795
X	16	1002
0	17	1374
3	17	1276
2	19	1631
1	20	1884

Optimum	**243**

Synergy Matrix for Problem : prob.42

Population-based Tier \ Local Search Tier	A	B	C	D
3	✓	✓	✓	✓
2		✓	✓	✓
1				

Table 17: Performance Analysis – Limited Time Trial on prob.42



Solver Dominance Matrix for Problem : prob.100

First Solver \ Second Solver	0	1	2	3	A	B	C	D	3A	1B	2B	3B	1C	2C	3C	1D	2D	3D	X	Y
Y	-	-	-	-	-	+	+	+	-	-	-	-	-	-	-	-	-	-		
X	-	-	-		+	+	+	+	+	-	-	-	-	-	-	-	-	-		+
3D	+		+	+	+	+	+	+	+										+	+
2D	+		+	+	+	+	+	+	+										+	+
1D	+		+	+	+	+	+	+	+										+	+
3C	+		+	+	+	+	+	+	+										+	+
2C	+		+	+	+	+	+	+	+										+	+
1C	+		+	+	+	+	+	+	+										+	+
3B	+		+	+	+	+	+	+	+										+	+
2B	+		+	+	+	+	+	+	+										+	+
1B	+		+	+	+	+	+	+	+										+	+
3A	-	-	-	-	-	+	+	+		-	-	-	-	-	-	-	-	-	-	+
D	-	-	-	-	-	+	-		-	-	-	-	-	-	-	-	-	-	-	-
C	-	-	-	-	-	+		+	-	-	-	-	-	-	-	-	-	-	-	-
B	-	-	-	-		-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
A	-	-	-	-		+	+	+	+	-	-	-	-	-	-	-	-	-	-	+
3	-	-	-		+	+	+	+	+	-	-	-	-	-	-	-	-			+
2	+	-		+	+	+	+	+	+	-	-	-	-	-	-	-	-	-	+	+
1	+		+	+	+	+	+	+	+										+	+
0		-	-	+	+	+	+	+	+	-	-	-	-	-	-	-	-	-	+	+

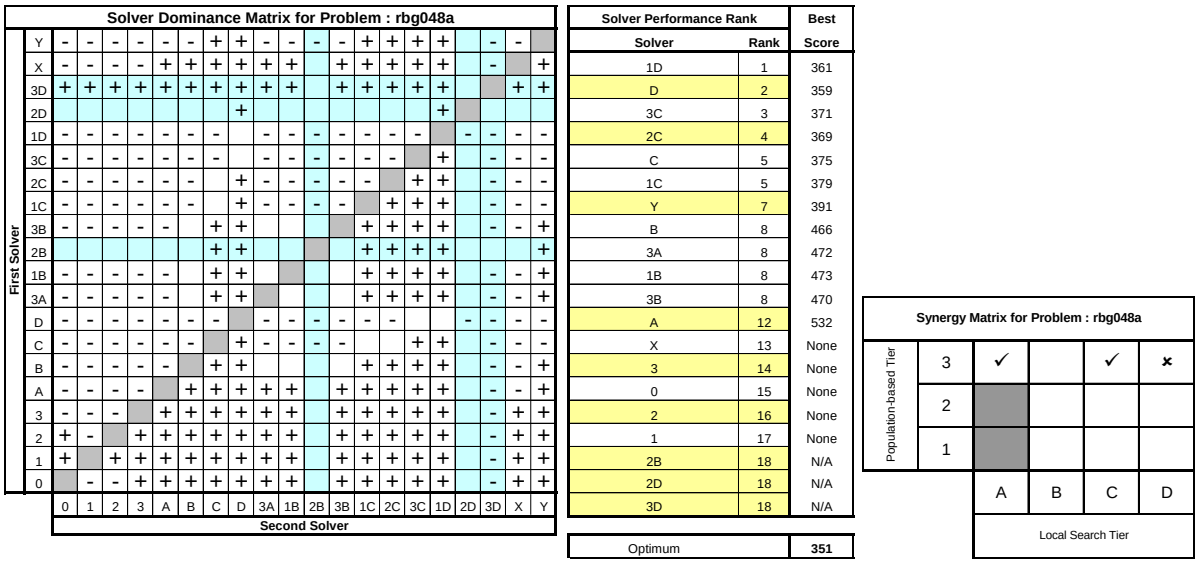
Solver Performance Rank		Best
Solver	Rank	Score
B	1	3009
D	2	3822
C	3	4376
Y	4	4396
3A	5	9961
A	6	12426
3	7	23137
X	7	21157
0	9	22299
2	10	23288
1	11	None
1B	11	N/A
2B	11	N/A
3B	11	N/A
1C	11	N/A
2C	11	N/A
3C	11	N/A
1D	11	N/A
2D	11	N/A
3D	11	N/A

Best Known Solution	**1385**

Synergy Matrix for Problem : prob.100

Population-based Tier \ Local Search Tier	A	B	C	D
3	✓	✗	✗	✗
2		✗	✗	✗
1				

Table 18: Performance Analysis – Limited Time Trial on prob.100

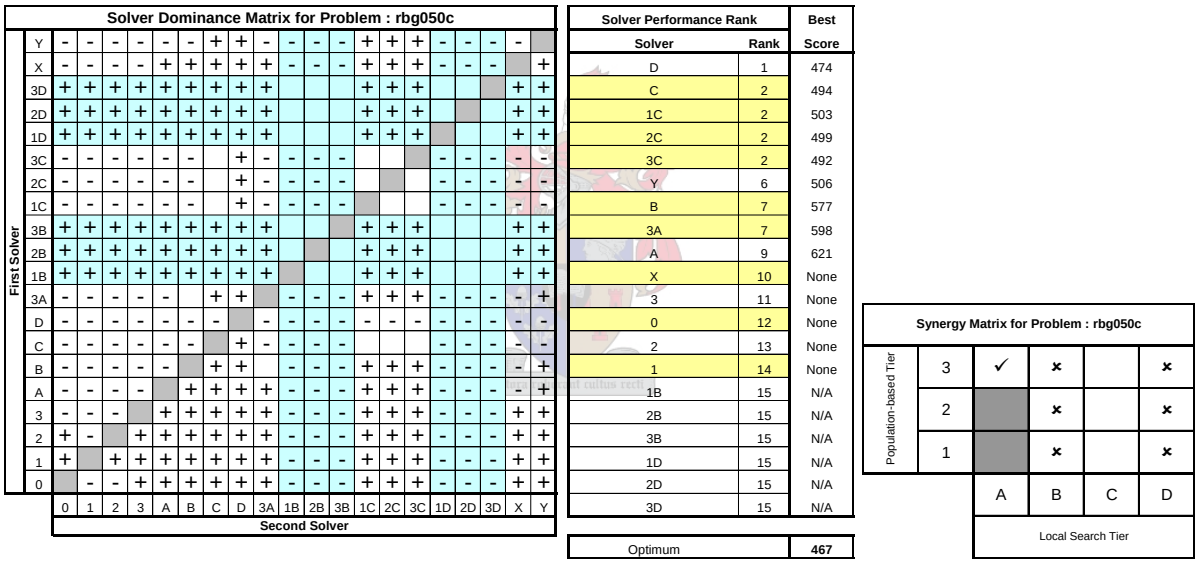


Solver Dominance Matrix for Problem : rbg048a																			
First Solver \ Second Solver	0	1	2	3	A	B	C	D	3A	1B	2B	3B	1C	2C	3C	1D	2D	3D	X	Y
Y	-	-	-	-	-	-	+	+	-	-	-	-	+	+	+	+		-	-	
X	-	-	-	-	+	+	+	+	+	+		+	+	+	+	+		-		+
3D	+	+	+	+	+	+	+	+	+	+		+	+	+	+	+			+	+
2D								+								+				
1D	-	-	-	-	-	-	-		-	-	-	-	-	-	-		-	-	-	-
3C	-	-	-	-	-	-	-		-	-	-	-	-	-		+		-	-	-
2C	-	-	-	-	-	-		+	-	-	-	-	-		+	+		-	-	-
1C	-	-	-	-	-	-		+	-	-	-	-		+	+	+		-	-	-
3B	-	-	-	-	-		+	+					+	+	+	+		-	-	+
2B							+	+					+	+	+	+				+
1B	-	-	-	-	-		+	+					+	+	+	+		-	-	+
3A	-	-	-	-	-		+	+					+	+	+	+		-	-	+
D	-	-	-	-	-	-	-		-	-	-	-	-	-			-	-	-	-
C	-	-	-	-	-	-		+	-	-	-	-	-		+	+		-	-	-
B	-	-	-	-	-		+	+					+	+	+	+		-	-	+
A	-	-	-	-		+	+	+	+	+		+	+	+	+	+		-	-	+
3	-	-	-		+	+	+	+	+	+		+	+	+	+	+		-	+	+
2	+	-		+	+	+	+	+	+	+		+	+	+	+	+		-	+	+
1	+		+	+	+	+	+	+	+	+		+	+	+	+	+		-	+	+
0		-	-	+	+	+	+	+	+	+		+	+	+	+	+		-	+	+

Solver Performance Rank		Best
Solver	Rank	Score
1D	1	361
D	2	359
3C	3	371
2C	4	369
C	5	375
1C	5	379
Y	7	391
B	8	466
3A	8	472
1B	8	473
3B	8	470
A	12	532
X	13	None
3	14	None
0	15	None
2	16	None
1	17	None
2B	18	N/A
2D	18	N/A
3D	18	N/A
Optimum		351

Synergy Matrix for Problem : rbg048a					
Population-based Tier	3	✓		✓	✖
	2				
	1				
		A	B	C	D
		Local Search Tier			

Table 19: Performance Analysis – Limited Time Trial on rbg048a



Solver Dominance Matrix for Problem : rbg050c																			
First Solver \ Second Solver	0	1	2	3	A	B	C	D	3A	1B	2B	3B	1C	2C	3C	1D	2D	3D	X	Y
Y	-	-	-	-	-	-	+	+	-	-	-	-	+	+	+	-	-	-		
X	-	-	-	-	+	+	+	+	+	-	-	-	+	+	+	-	-	-		+
3D	+	+	+	+	+	+	+	+	+				+	+	+				+	+
2D	+	+	+	+	+	+	+	+	+				+	+	+				+	+
1D	+	+	+	+	+	+	+	+	+				+	+	+				+	+
3C	-	-	-	-	-	-		+	-	-	-	-				-	-	-	-	-
2C	-	-	-	-	-	-		+	-	-	-	-				-	-	-	-	-
1C	-	-	-	-	-	-		+	-	-	-	-				-	-	-	-	-
3B	+	+	+	+	+	+	+	+	+				+	+	+				+	+
2B	+	+	+	+	+	+	+	+	+				+	+	+				+	+
1B	+	+	+	+	+	+	+	+	+				+	+	+				+	+
3A	-	-	-	-	-		+	+		-	-	-	+	+	+	-	-	-	-	+
D	-	-	-	-	-	-	-		-	-	-	-	-	-	-	-	-	-	-	-
C	-	-	-	-	-	-		+	-	-	-	-				-	-	-	-	-
B	-	-	-	-	-		+	+		-	-	-	+	+	+	-	-	-	-	+
A	-	-	-	-		+	+	+	+	-	-	-	+	+	+	-	-	-	-	+
3	-	-	-		+	+	+	+	+	-	-	-	+	+	+	-	-	-	+	+
2	+	-		+	+	+	+	+	+	-	-	-	+	+	+	-	-	-	+	+
1	+		+	+	+	+	+	+	+	-	-	-	+	+	+	-	-	-	+	+
0		-	-	+	+	+	+	+	+	-	-	-	+	+	+	-	-	-	+	+

Solver Performance Rank		Best
Solver	Rank	Score
D	1	474
C	2	494
1C	2	503
2C	2	499
3C	2	492
Y	6	506
B	7	577
3A	7	598
A	9	621
X	10	None
3	11	None
0	12	None
2	13	None
1	14	None
1B	15	N/A
2B	15	N/A
3B	15	N/A
1D	15	N/A
2D	15	N/A
3D	15	N/A
Optimum		467

Synergy Matrix for Problem : rbg050c					
Population-based Tier	3	✓	✖		✖
	2		✖		✖
	1		✖		✖
		A	B	C	D
		Local Search Tier			

Table 20: Performance Analysis – Limited Time Trial on rbg050c

In the first two problems, br17.12 and ESC12, the majority of the solvers identified a global optimum within the time period and consequently; these solvers' performance could not be differentiated on these problems.

For three of the problems, prob.100, rbg048a and rbg050c, some of the solvers failed to complete a single iteration within the designated time period. These solvers automatically failed their synergy tests.

5.5 SOLVER PERFORMANCE WITH EXTENDED RUN TIMES

In this section, the best performing solvers are evaluated on two of the most challenging problems in the benchmark set over an extended run time of 5 hours. The two benchmark problems were also evaluated for comparison.

The two tiered solvers 3D and 3B were identified as the best performing solver in the limited time trials described in the previous section and they were selected for the experiment in this section.

Benchmark problems rbg048a and rbg050c are two of the most challenging benchmark problems for which the objective function value of the global optimum is known. They are characterised by about 50 locations and at least four times as many constraints as any of the other benchmark problems. These problems were selected for the experiment in this section.

For this experiment, each solver was allowed three attempts to solve each of the benchmark problems. The results are shown in Table 21 and Table 22 and optimal runs are shown in grey:

Solver	Attempt	Run Time (hours)	Value	Optimum
2D	1	0.90	351	Yes
2D	2	5.00	353	No
2D	3	5.00	352	No
3D	1	1.94	351	Yes
3D	2	1.47	351	Yes
3D	3	1.46	351	Yes
X	1	5.00	Infeasible	No
X	2	5.00	Infeasible	No
X	3	5.00	Infeasible	No
Y	1	5.00	453	No
Y	2	5.00	434	No
Y	3	5.00	419	No

Table 21: Extended run results for rbg048a

Solver	Attempt	Run Time (hours)	Value	Optimum
2D	1	5.00	472	No
2D	2	5.00	471	No
2D	3	5.00	472	No
3D	1	0.77	467	Yes
3D	2	5.00	471	No
3D	3	5.00	469	No
X	1	5.00	Infeasible	No
X	2	5.00	Infeasible	No
X	3	5.00	Infeasible	No
Y	1	5.00	509	No
Y	2	5.00	541	No
Y	3	5.00	581	No

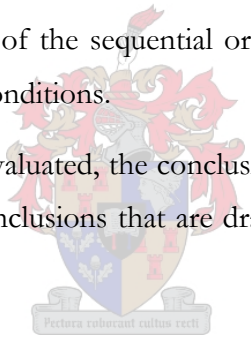
Table 22: Extended run results for rbg050c

The following observations can be made from these results:

- Solver 3D was able to identify global optima for rbg048a in less than 2 hours with every repetition and it was the only solver to identify an optimum for rbg050a, although it managed to do so only once.
- Solver 2D managed to identify a global optimum only once for rbg048a, but managed to come within one or two units of the optimum in the remaining runs. For rbg050c, every run managed to come within five units of the optimum in every repetition.
- Benchmark method X, the GA with edge assembly crossover, was unable to find any feasible solutions to either of the benchmark problems within the 5 hours trials.
- Benchmark method Y, the simulated annealer was unable to find any global optima for any of the benchmark problems and performed considerably worse than both the two-tiered solvers.

This experiment was done to confirm that two-tiered solvers are capable, if given sufficient time, of solving very complex instances of the sequential ordering problem, and can outperform the benchmark methods under these conditions.

Since only three repetitions were evaluated, the conclusions that can be drawn from this data are limited, but it does support the conclusions that are drawn in Chapter 6, from the results in the previous two sections.



5.6 CHAPTER SUMMARY

In this chapter, the solvers developed during the course of this project are evaluated according to their performance on benchmark sequential ordering problems.

In the first section, the parameters of the experiment are stated. These include the benchmark problems that are used, the number of trials or repetitions that are evaluated for each problem-solver pair and the solver specific parameters that are used to control solver execution.

In Section 5.2, the time complexity of the solvers is analysed to estimate the growth in the time required to perform a single iteration of the solver as a function of the problem size. For most of the solvers, the increase in time as a function of the n, problem size, seems to grow at $O(n^k)$, where $k<5$, this means that the growth rate can be bounded by a polynomial function.

In Section 5.3 results are presented for cases where some of the solvers managed to locate global optima. These results were only achieved on three of the ten benchmark problems. In each instance, the time required to locate the global optimum was recorded for each trial and the results were compared using the Wilcoxon rank-sum test, a non-parametric method that is used to test whether the medians of two distributions are equal. The results of the analysis are summarised using three visual aids: a dominance matrix, which indicates which solvers perform significantly better or worse than others, a ranked list of solver performance and a synergy matrix which indicates whether or not each two-tiered solver outperforms its component tiers.

In Section 5.4, the performances of the solvers are evaluated in limited time trials on the benchmark problems. The best candidate solution was recorded for each trail and the results were compared and presented using the same analysis technique and visual aids as those used in Section 5.3. During this analysis solvers are identified that were unable to return any candidate solutions within the prescribed time period.

In the following chapter, conclusions will be drawn from the analyses in this chapter, recommendations will be made to improve the performance of two-tiered solvers and opportunities for further research will be identified.

6 CONCLUSIONS AND RECOMMENDATIONS

In this chapter conclusions drawn from the performance analyses in the previous chapter will be given. Also, recommendations for further improvements will be made and opportunities for further research identified.

The conclusions in this chapter will indicate that the two-tiered solver architecture developed in this project performed satisfactorily by enabling two-tiered solvers that significantly outperform their components tiers. Furthermore, these solvers have proven to be superior to at least two competing methods that are used in practice to solve combinatorial optimisation problems like the sequential ordering problem.

The first section of this chapter states the conclusions that were drawn from the analyses of solver performances referred to in the first chapter. These conclusions relate to:

- the effectiveness of the two-tiered architecture in improving on the performance of each one's component tiers,
- the performance of two-tiered solvers in comparison to benchmark methods used in practice,
- the impact of the symmetry of the transition cost matrix of a problem have on solver performance, and
- the computational complexity of two-tiered solvers.

In the second section, recommendations will be made on the use and further improvement of the two-tiered solvers that were developed in this project, and opportunities for further research will be identified.

The last section of this chapter contains a summary of the project.

6.1 CONCLUSIONS

The conclusions in this subsection are based only on the analyses reported in the previous chapter. Both the time required to reach global optima and the results of the limited time trials were considered in these conclusions.

Some questions remain that were either not addressed during this study, or for which the analysis in the previous chapter is not sufficient to substantiate anything more. Such questions are identified as potential areas for further research in the next section.

6.1.1 The effectiveness of two-tiered solvers

A key deliverable for this project is the development of a two-tiered architecture for solving the sequential ordering problem and to quantify the performance of two-tiered solvers in relation to their component tiers. First, the performances of the component tiers are considered relative to one-another and then the benefit of the interaction between two different tiers in a two-tiered solver is considered.

a.) Component tier performances

The two-tiered solvers developed in this project consist of a population-based tier, whose function it is to explore and maintain knowledge about a problem's search space, this function is achieved by maintaining a diverse population of candidate solutions. Three population-based tiers were implemented:

1. A particle swarm optimiser that uses 2-opt neighbourhood transitions to move candidate solutions through the problem's search space.
2. A genetic algorithm that uses 2-opt transitions between parent solutions to produce offspring.
3. A genetic algorithm that uses partially mapped crossover (see Subsection 4.2.3 on page 82) between parent solutions to produce offspring.

The second tier in the two-tiered architecture is a local search tier, whose role it is to exploit promising regions of a problem's search space by performing steepest-descent local search using a specific neighbourhood relation. The following local search methods were implemented for this project:

1. A 2-opt local search.
2. A local search method that inverts sections of the candidate solution sequences.

3. A local search method that uses pair-wise exchange of elements in candidate solution sequences.

4. A local search method that displaces single elements within candidate solution sequences.

The first significant conclusion is that repeated local search using any of the local search methods yields significantly and consistently better results than any of the population-based methods. The only instances where repeated local searches do not significantly outperform the population-based searches are where these approaches fail to find optimal solutions within a prescribed time limit on ESC25 (see Section 5.3 on page 105).

In general, the performance of the population-based methods was poor and in the majority of the results, population-based methods could not outperform repeated random guessing. The only exceptions are on problems ft70.1, prob.100, rbg048a and rbg050c, where the genetic algorithm using partially mapped crossover performed significantly better that repeated random guessing

b.) Two-tier solver performance

The success of a two-tiered solver can be measured by the ability of the solver to outperform its component tiers. When a two-tiered solver manages to significantly outperform both its component tiers, the two-tiered architecture has succeeded in creating synergy and contributes to the overall performance of the solver. The results in Chapter 5 show that the synergy between the tiers is strongly dependent on the choice of the population-based tier.

When drawing conclusions on the performance of two tiered solvers, it needs to be considered that on problems prob.100, rbg048a and rbg050c, some instances of two-tiered solvers were unable to return solutions within the time limit and in these cases no meaningful conclusions could be drawn with regards to synergy.

The genetic algorithm using partially mapped crossover is the most successful in creating synergy and thereby improving the performance of a two-tiered solver beyond that of its components. Two-tiered solvers that use this tier never performed significantly worse than their strongest component. This population-based tier failed to improve on its components in the following instances and for the following reasons:

- On problems br17.1 and ESC12, the majority of solvers managed to find global optima and therefore, the solutions from the two-tiered solvers using this population-based method could not improve on the optimal solutions found by the local search tiers. However, when the time required to locate global optima was measured, all of the two-tiered methods exhibited synergy.

The genetic algorithms using 2-opt exchanges between parents to produce offspring never performed significantly worse than its component tiers. In the majority of the benchmark problems, two-tiered solvers using this population-based tier outperformed their individual tiers.

As a population-based tier, the implementation of a particle swarm optimiser used in this project did not succeed in creating synergy in a two-tiered solver. No instances of two-tiered solvers using this population-based tier significantly outperformed their local search tier and some of these two-tiered solvers performed significantly worse than their local search tiers. This implementation failed to add any value to finding solutions to the benchmark problems.

Since the population-based tiers on their own are not strong performers when applied to the benchmark problems, the performance benefit that they bring to the two-tiered solvers is more likely to stem from their ability to maintain a diverse set of representative solutions, than from their ability to optimise individual solutions in their populations. In a two-tiered solver the optimisation task seems to be performed primarily on the local search tiers.

6.1.2 Local search tier performance and the significance of problem symmetry

From the results in Chapter 5, it appears that if a two two-tiered solver are constructed using the same population-based tier, then the better performing local search tier, will also yield the better performing two-tiered solver.

This means a ranking of the local search tiers according to their performance will also extend to the ranking of the two-tiered solvers of which they form part.

On the benchmark problems, the following performance was observed for the local search tiers: If the problem's transition cost matrix is symmetrical, the local search tier that uses the substring inversion neighbourhood relation outperforms other local search tiers; for all other problems, either the local search tier that uses the substring inversion neighbourhood relation, or the local search tier that displaces single elements in candidate solution sequences, provides the best local search tier performance.

The two most successful solvers evaluated in the course of this project were 3B and 3D. Both used the genetic algorithm with partially mapped crossover as the population-based component. The local search components used the substring inversion neighbourhood relation and the single element displacement relation respectively.

6.1.3 The performance of benchmark solvers

Two alternative solvers were used as benchmarks for evaluating the two-tiered solvers developed in this project. In this subsection, the performance of these methods is compared to that of the other solvers.

a.) The genetic algorithm using edge assembly crossover (EAX)

The genetic algorithm using edge assembly crossover is a population-based method that has been proven to be successful at addressing sequencing problems like the travelling salesman problem. It was included in this project as a benchmark to compare to the two-tiered solvers developed in the course of the project.

This method performed well. It mostly outperformed and never performed significantly worse than any of the population-based tiers. However this method was never able to outperform any of the local search tiers or any of the two-tiered solvers.

b.) The simulated annealing algorithm

The simulated annealing algorithm uses pair-wise exchange to determine suitable transitions. This method's performance is closely linked to the performance of the local search tier that uses a pair-wise exchange neighbourhood relation.

When compared to the performance of the local search tier that uses a pair-wise exchange neighbourhood relation, the simulated annealing algorithm never outperformed this local search tier or any two-tiered algorithm of which the local search tier was a part.

This algorithm did manage to improve on some of the local search tiers and two-tiered solvers, provided that these methods did not outperform a local search tier that uses pair-wise exchange.

When comparing the two benchmark methods to the two-tiered solvers developed in this project, the two-tiered solvers and at least some of the local search tiers on their own, performed significantly better that those of the benchmark methods.

On all the benchmark problems, some of the two-tiered solvers were consistently able to outperform both benchmark methods.

6.1.4 Runtime requirements of two-tiered solvers

The time required by each solver to complete a single iteration was measured. In this analysis the time was measured as a function of the size of the problem, which is measured by the number of elements in its solution sequence. To minimise the impact of other factors, such as the number of constraints, only problems with less than 20 constraints were used in this analysis.

The following observations can be made about the runtime requirements of all the solvers:

The time required to complete an iteration on most of the solvers can be bound by a polynomial function of the problem's size with an order of less than 5, where the size of the problem is measured by the number of elements in its solution sequences.

Since the simulated annealing algorithm was implemented to run iterations within a fixed time period, this method is mostly unaffected by the size of the problem. However, for large problems, this method might fail to arrive at any solution within a prescribed time limit.

The genetic algorithm using partially mapped crossovers displayed sub-linear growth in the time required to iterate as the problem size increases.

For some of the solvers, most notably three of the two-tiered solvers based on the genetic algorithm using partially mapped crossover, a linear model could not be assumed based on the available data. Therefore no further conclusions were drawn on the runtime requirements of these solvers.

6.1.5 Project objectives

The deliverables for this project have been articulated as follows:

- Develop a new method to solve or approximate solutions to sequential ordering problems. This method needs to separate exploitative behaviour that makes incremental improvements through "greedy" decisions, from exploratory behaviour that seeks out promising new areas in the solution space.
- Quantify the benefit or handicap that results from combining exploratory and exploitative behaviour by evaluating the different behaviour of the newly developed method.
- Benchmark the new method against competing methods.

The conclusions in this section confirm that these objectives have been achieved.

The two-tiered solver architecture succeeded in combining population-based and local search heuristics in a way that yielded better performance than these heuristics on their own.

When compared to benchmark methods to address the sequential ordering problem, the two-tiered architecture produced competitive solvers that significantly outperformed the two benchmark methods chosen for this project.

6.2 RECOMMENDATIONS

Through this project, a two-tiered solver architecture was developed that has been proven to be competitive in providing solutions for the sequential ordering problem. Although the success of this method has been proven, many opportunities remain to improve performance and further investigate this direction of research. These opportunities are presented here.

6.2.1 Recommended actions to improve solver performance

Since the two-tiered architecture has proven to be effective in providing solutions to the sequential ordering problem, the use of two-tiered solvers can be recommended for addressing problems of this nature. In particular, solvers 3B and 3D were the most successful on benchmark problems. The first solver uses the subsequence inversion as a local search tier and a genetic algorithm with partially mapped crossover as the population-based tier and the second solver uses the single element displacement as a local search with the same genetic algorithm as the population-based tier.

Three actions have been identified to improve the performance of two-tiered solvers, a) improved implementation, b) parameter optimisation and c) using the genetic algorithms with edge assembly crossover as a population-based tier.

a.) Improved solver implementation

The implementation of all the solvers in this project was coded in Java. While Java is an efficient and convenient language for development, the Java language is executed on a so-called virtual machine. Typically, Java programs are executed in an interpreted environment and executes more slowly than languages that compile directly into native machine code. Significant improvements in solver performance may be realised by coding these solvers in a language like C++ that compiles directly to machine code. Furthermore, the implementations used in this project have not been optimised to ensure that the most efficient code is used for achieving the desired behaviour. Code optimisation could also present a significant opportunity to improve solver performance.

b.) Parameter optimisation

Several of the two-tiered solvers have different parameters that influence their effectiveness. These parameters include:

- The population size used in population-based tiers,
- The probabilities of crossover and mutation in the genetic algorithms that were used in population-based tiers.

- The probability of selection that was of the genetic algorithms that formed population-based tiers.

During this project, little attention was given to the selection of these parameters. The optimisation of these parameters may offer additional opportunities to improve the performance of the two-tiered solvers. Optimal parameters could be dependent on problem characteristics such as: the problem size, the number of constraints that a problem has or whether or not the transition cost matrix of the problem is symmetrical.

The rank-space selection method applied in all the genetic algorithms of the two-tiered solvers give equal recognition to the performance and the diversity ranks of solutions. It is possible to vary the weight attached to one rank or the other to influence the performance of a solver. The influence of these parameters on solver performance is not well understood and more investigation is required to improve in this area.

c.) Use the genetic algorithms with edge assembly crossovers as a population-based tier

Finally, the genetic algorithm using edge assembly crossover was tested as a benchmark method, but was not evaluated as a candidate for the population-based tiers of two-tiered solvers. Since this method was the best performing population-based method, apart from the two-tiered solvers, this option is promising and should be investigated.

6.2.2 Further research opportunities

In this subsection further research opportunities are identified that build on the two-tiered solvers developed in this project. These opportunities are:

- Further investigation of run-times required to identify global optima.
- Developing a better understanding of the time required to perform a single solver iteration as a function of problem size.
- Investigating the impact of the number and structure of constraints on the time required to perform a single solver iteration.
- Investigating the distribution of candidate solutions in two-tiered solvers and how the distribution changes over iterations.
- Developing the application of population-based tiered algorithms in multi-objective optimisation.
- Integration of the exploratory and exploitative behaviour found into a single tiered solver, without compromising performance.
- Developing the application of two-tiered solvers to other combinatorial optimisation problems.

a.) Further investigation of run-times required to identify global optima

In this project, optimal solutions were only found for three of the ten benchmark problems. A limiting factor in finding optimal solutions for other problems was the time limit of 300 seconds per trial.

Future projects could expand on the results found in this project by increasing the 300 second time limit to identify local optima on more difficult sequential ordering problems.

By understanding the time required to identify a global optimum, a decision maker would be better equipped when deciding whether to search for an optimum, or settle for an approximate solution. This information would also give a decision maker a better idea of when to stop a search, since in practice, there is often no way of confirming whether or not an optimal solution has been identified.

b.) Developing a better understanding of the time required per solver iteration as a function of problem size

In this project, the time required to perform solver iterations as a function of the problem size was analysed through linear regression.

There is an opportunity, through theoretical and practical analysis, to gain better understanding of how the time per iteration of a solver increases with the problem size.

This knowledge could be useful in selecting the most appropriate solver for particular problems.

c.) Investigating the impact of the number and structure of constraints on the time required per solver iteration

In this project, the time per solver iteration was measured as a function of problem size, without considering the number and structure of the precedence constraints.

Further research in this area could improve the understanding of what influence constraints have on a solver's iteration time.

This knowledge could be useful in selecting the most appropriate solver given a problem's constraints.

d.) Investigating the impact of the number and structure of constraints on the time required per solver iteration

The principles that the two-tiered solvers are built on require that a solver will maintain and constantly improve on a diverse set of locally optimal solutions.

Detailed investigation of the changes in the population over the iterations of a two-tiered solver could confirm whether or not this principle is valid.

A better detailed understanding of how the population of a two-tiered solver changes over time could be used to improve the performance of the solver.

e.) Develop the application of population-based tiered algorithms in multi-objective optimisation.

By maintaining multiple, diverse, locally optimal solutions, the two-tiered solver gives a decision maker a number of near optimal candidate solutions to choose from.

It may be possible to adapt the two-tiered architecture so that the population is driven towards a set of solutions that simultaneously seeks to optimise more than one objective function.

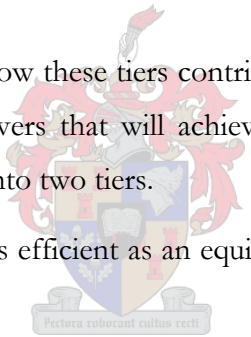
Such an architecture could be used to approximate Pareto-optimal solution sets to multiple-objective optimisation problems. Some preliminary investigation will be required to determine whether this is a viable research topic.

f.) Integration of the exploratory and exploitative behaviour found into a single tiered solver, without compromising performance.

The two-tiered solvers developed in this project explicitly separated exploitative and exploratory behaviour into two separate tiers.

Through better understanding of how these tiers contribute to the overall function of the solver, it may be possible to develop solvers that will achieve similar performance without explicitly separating the required behaviour into two tiers.

Such a solver needs to be at least as efficient as an equivalent two-tiered solver to be of practical use.



g.) Develop the application of two-tiered solvers to other combinatorial optimisation problems.

Finally, although this project was concerned with the sequential ordering problem, the two-tiered approach should be easy to adapt to solve many other combinatorial optimisation problem.

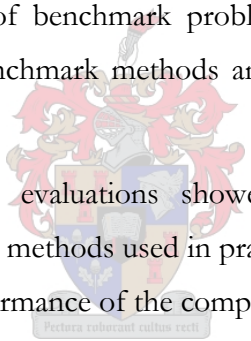
6.3 PROJECT SUMMARY

This project developed new solvers using a systematic approach. First the structure of the sequential ordering problem was investigated, then techniques were researched to address the sequential ordering problem and other related problems. The objectives for the newly developed solvers were formulated and operating principles were identified to achieve these objectives. Only when the principles were clear, was a structure specified and were solvers implemented according to this structure.

The projects' primary objective was to develop a new class of solver for the sequential ordering problem that explicitly separates exploratory and exploitative behaviours into two separate tiers.

An architecture for these solvers was developed based on the objectives of the solver. Operating principles were identified to address these objectives and finally a structure was specified to support and enable the operating principles.

A number of solvers were developed according to the structure. The performances of these solvers were evaluated on a set of benchmark problems. They were measured against their individual component tier, the benchmark methods and one another in solving the sequential ordering problem.



The results of the performance evaluations showed that these two-tiered solvers were competitive in comparison to other methods used in practice, and that the two-tiered architecture is successful in improving the performance of the component tiers.

Although the two-tiered solvers were successfully developed and implemented, many opportunities remain to further improve their performance and develop a better understanding of how they work and where they can be applied. These opportunities were identified and documented earlier in this chapter.

REFERENCES

Abido, M. A. (2002a). "Optimal design of power system stabilisers using particle swarm optimisation". IEEE Transactions on Energy Conservation, vol. 17 (3), pp. 406-413.

Antonisse, J. H. (1989). "A new interpretation of schema notation that overturns the binary encoding constraint". In Proceedings of the Third International Conference on Genetic Algorithms, pp. 86-91.

Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., Protasi, M. (1999). "Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties", Springer-Verlag, Heidelberg, Germany.

Baker, J. E., (1985), "Adaptive selection methods for genetic algorithms", In Proceedings of the First International Conference on Genetic Algorithms and their Applications, Earlbaum.

Clerq, M., (2000), "Discrete particle swarm optimisation illustrated by the travelling salesman problem", Technical Report, http://clerc.maurice.free.fr/pso/

Cockshott, A. R., & Hartman, B. E. (2001). "Improving the fermentation medium for Echinocandin B production part II: Particle swarm optimisation". Process Biochemistry, vol. 36, pp. 661-669.

Conradie, A., Miikkulainen, R., & Aldrich, C. (2002). "Intelligent process control utilizing symbiotic memetic neuro-evolution". Proceedings of the 2002 Congress on Evolutionary Computation, pp. 623-628.

Cook, S. A. (1971). "The complexity of theorem-proving procedures", Proceedings of the 3rd ACM Symposium on Theory of Computing, ACM, pp. 151-158.

Davis, L., Editor. (1991), "Handbook of Genetic Algorithms", Van Nostrand Reinhold, New York.

De Jongh, K. A. (1975). "An Analysis of the Behaviour of a Class of Genetic Adaptive Systems". (Doctoral dissertation, University of Michigan), Dissertation Abstract International, 36(10), 5140B.

De Jongh, K. A. (1993). "Genetic Algorithms are NOT Function Optimizers". In Foundations of Genetic Algorithms 2, Morgan Kaufmann Publishers, San Mateo, California, pp. 5-17.

De la Maza, M., & Tidor, B. (1991). "Increase flexibility in genetic algorithms: The use of variable Boltzmann selective pressure to control propagation". Proceedings of the ORSA CSTS

Conference - Computer Science and Operations Research: New Developments in their Interfaces, pp. 425-440.

Eberhart, R. C., & Kennedy J. (1995). "A new optimiser using particle swarm theory". Proceedings of the Sixth International Symposium on Micromachine and Human Science, pp. 39-43.

Eberhart, R. C. & Shi, Y. (1998). "Evolving artificial neural networks". Proceedings of International Conference on Neural Networks and Brain, Beijing. pp. PL5-PL13

El-Gallad, A. I., El-Hawari, M. E., Sallam, A. A., Kalas, A. (2001). "Swarm-intelligently trained neural network for power transformer protection". Canadian Conference on Electrical and Computer Engineering, pp. 265-269.

El-Mounayri, H., Dulga, Z., & Deng, H. (2003). "Predicting surface roughness in end milling using particle swarm optimisation". Proceedings of the Swarm Intelligence Symposium 2003 (SIS 2003, Indianapolis, Indiana), pp. 220-227.

Escudero, L. F., (1988). "An inexact algorithm for the sequential ordering problem", European Journal of Operational Research, vol. 37, pp. 232-253.

Forrest, S., & Mitchell, M. (1993). "Relative Building Block Fitness and the Building Block Hypothesis". In Foundations of Genetic Algorithms 2, Morgan Kaufmann Publishers, San Mateo, California, pp. 109-126.

Fraser, A. S., (1957a). "Simulation of genetic systems by automatic digital computers I: Introduction". Australian Journal of Biological Science, vol. 10, pp. 484-491.

Fraser, A. S., (1957b). "Simulation of genetic systems by automatic digital computers II: Effects of linkage on rates of advance under selection". Australian Journal of Biological Science, vol. 10, pp. 492-499.

Fukuyama, Y., Takayama, S., Nakanishi, Y., Yoshida, H. (1999). "A particle swarm optimization for reactive power and voltage control in electric power systems". Proceedings of the Genetic and Evolutionary Computation Conference 1999 (GECCO 1999), pp. 1523-1528.

Glover, F. & Knox, J. (1998), "Tabu Search: An effective heuristic for combinatorial optimization problems." Proceedings of the Third Annual Rocky Mountain Conference on Artificial Intelligence, pp. 306-323.

Goldberg, D. E. (1987). "Simple genetic algorithms and the minimal deceptive problem" In Davis, L. Ed., Genetic Algorithms and Simulated Annealing, Pitman, London, pp.74-88.

Goldberg, D. E. (1989a). "Genetic algorithms in search optimization and machine learning" Addison-Wesley, Reading.

Goldberg, D. E. (1989b). "Sizing populations for serial and parallel genetic algorithms" In Proceedings of the Third International Conference on Genetic Algorithms, pp. 70-79.

Goldberg, D. E. & Deb, K. (1991). "A comparative analysis of selection schemes used in genetic algorithms" In Foundations of Genetic Algorithms, Morgan Kaufmann Publishers, San Mateo, California, pp. 69-93.

Grefenstette, J. J. (1993). "Deception Considered Harmful". In Foundations of Genetic Algorithms 2, Morgan Kaufmann Publishers, San Mateo, California, pp. 75-91.

Gudise, V. G., & Venayagamoorthy, G. K. (2003a). "Comparison of particle swarm optimization and back propagation as training algorithms for neural networks". Proceedings of the IEEE Swarm Intelligence Symposium 2003 (SIS 2003), pp. 110-117.

Holland, J. (1975). "Adaptation in Natural and Artificial Systems", The MIT Press, Cambridge, Massachusetts.

Ismail, A., & Engelbrecht A.P. (1999). "Training Product Units in Feedforward Neural Networks using Particle Swarm Optimization". Proceedings of the International Conference on Artificial Intelligence, pp. 34-40.

Ismail, A., & Engelbrecht, A. P. (2000). "Global optimisation algorithms for training product unit neural networks". Proceedings of IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN 2000), pp. 132-137.

Kennedy, J., & Eberhart, R. C. (1997). "A discrete binary version of the particle swarm algorithm". Proceedings of the World Multiconference on Systemics, pp. 4104-4109.

Kirkpatrick, S., Gelatt C. D. & Vecchi M. P. (1983). "Optimization by Simulated Annealing", Science, Vol 220, Number 4598, pp. 671-680.

Laskari, E. C., Parsopoulos, K. E. & Vrahatis, M. N. (2002) "Particle swarm optimization for integer programming". Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2002), pp. 1582-1578.

Mendes, R., Cortez, P., Rocha, M., Neves, J. (2002). "Particle swarms for neural network training". Proceedings of the 2002 Joint Conference on Neural Networks (IJCNN 2002), pp. 1895-1899.

Michalewicz, Z. (1992). "Genetic Algorithms + Data Structures = Evolution Programs", Springer-Verlag, New York.

Michalewicz, Z., Fogel, D.B. (2002). "How to solve it: Modern Heuristics", Springer-Verlag, New York.

Mitchell, M. (1996), "An Introduction to Genetic Algorithms", The MIT Press, Cambridge, Massachusetts.

Mitchell, M., Forrest, S., & Holland, J. H. (1992). "The Royal Road for genetic algorithms: fitness landscapes and GA performance." In Varela, F. J. & Bourgine, P., eds., Advances in Neural Information Processing Systems 6, Morgan Kaufmann.

Mocato, P. & Cotta, C. (2002). "A gentle introduction to memetic algorithms" In Glover, F. & Kochenberger, G. A., eds., Handbook of Metaheuristics, Kluwer Academic Publishers.

Mohan, C. K. & Al-Kazemi, B. (2001). "Discrete particle swarm optimization". Proceedings of the Workshop on Particle Swarm Optimiazation 2001. Indianapolis.

Nagata, Y., & Kobayashi, S. (1997). "Edge Assembly Crossover: A High-power Genetic Algorithm for the Travelling Salesman Algorithms". Proceedings of the Seventh International Conference on Genetic Algorithms, pp. 450-457.

Orponen, P., & Mannila, H. (1987). "On approximation preserving reductions: Complete problems and robust measures", Technical Report C-1987-28, Department of Computer Science, University of Helsinki. (ND32, MP1, LO7).

Pullyblank, W. & Timlin, M., (1991). "Precedence constrained routing and helicopter scheduling: heuristic design", IBM Technical Report RC17154 (#76032).

Reeves, C. R., & Wright, C. C. (1995). "Epistasis in Genetic Algorithms: An Experimental Design Perspective". In Proceedings of the Sixth International Conference in Genetic Algorithms, pp. 217-224.

Salerno, J. (1997). "Using the particle swarm optimization technique to train a recurrent neural model". IEEE International Conference on Tools with Artificial Intelligence, pp. 45-49.

Salman, A., Ahmad, I., & Al-Madani, S. (2002). "Particle swarm optimization for task assignment problems". Microprocessors and Microsystems, vol. 26 (8), pp 363-371.

Savelsbergh, M. W. P., (1990). "An efficient implementation of local search algorithms for constrained routing problems", European Journal of Operations Research, vol. 35, pp. 254-265.

Secrest, B. R. (2001). "Travelling salesman problem for surveillance mission planning using particle swarm optimisation". Masters Thesis, School of Engineering and Management of the Air Force Institute of Technology, Air University.

Shi, Y., & Eberhart, R. C. (1998). "A modified particle swarm optimiser". Proceedings of the IEEE International Congress on Evolutionary Computation (CEC 1998), pp. 69-73.

Tagawa, K., Kanzaki, Y., Okada, D., Inoue, K. & Haneda, H. (1998). "A New Metric Function and Harmonic Crossover for Symmetric and Asymmetric Travelling Salesman Problems". The 1998 IEEE International Conference on Evolutionary Computation, pp. 822-827.

Tandon, V. E., H., & Kishawy, H. (2002). "NC end milling optimization using evolutionary computation". International Journal of Machine Tools and Manufacture, vol. 42 (5), pp. 595-605.

Voss, M. S., & Feng, X. (2002). "A new methodology for emergent system identification using particle swarm optimization (PSO) and the group method data handling (GMDH)". Proceedings of the Genetic and Evolutionary Computation Conference 2002 (GECCO 2002), pp. 1227-1232.

Walpole, R.E., Myers, R.H. (1990). "Probability and statistics for scientist and engineers", Maxwell-MacMillan, New York.

Winston, P.H. (1993). "Artificial Intelligence: 3rd Edition", Addison-Wesley, Reading, Massachusetts.

Wright, A.H. (1991), "Genetic algorithms for real parameter optimization", In First Workshop on the Foundations of Genetic Algorithms and Classifier Systems, Morgan Kaufmann, San Mateo, California, pp. 193-203.

Wu, A. S., Lindsay, R. K., & Riolo, R. L. (1997). "Empirical observations on the roles of crossover and mutation". Proceedings of the Seventh International Conference on Genetic Algorithms, pp. 362-369.