Meta-heuristic algorithms development for combinatorial optimization within an integrated problem solving environment

Xu, Yiliang

2007

Xu, Y. (2007). Meta-heuristic algorithms development for combinatorial optimization within an integrated problem solving environment. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/3822>

<https://doi.org/10.32657/10356/3822>

Nanyang Technological University

Downloaded on 26 Apr 2023 18:40:43 SGT

Meta-Heuristic Algorithms Development for Combinatorial Optimization within an Integrated Problem Solving Environment

Yiliang Xu

School of Electrical & Electronic Engineering

A thesis submitted to the Nanyang Technological University
in fulfillment of the requirement for the degree of
Doctor of Philosophy

2007

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research and has not been submitted for a higher degree to any other University or Institution.

.....
Date

.....
Yiliang Xu

Acknowledgement

I would like to express my sincere appreciation to the administration of Nanyang Technological University of Singapore for giving me the opportunity to complete my Ph.D. study. In particular I acknowledge the faculty from Intelligent System Centre (IntelliSys) for their kindly support.

I would like to thank my supervisor, Dr Meng-Hiot Lim, for years of professional guidance and supervision. In addition to his knowledge, experience and efforts, I am grateful for his understand and cares.

I would like to thank the anonymous reviewers for their insightful comments and suggestions for improving the quality of this thesis.

Special thanks also go to my parents and friends. Thanks for their support and encouragement. I also like to thank my lab mates from Intelligent System Center for providing friendly environment.

Summary

Many practical real-world combinatorial optimization problems tend to be computationally intractable. While advancement in computer technology is able to provide significant headway by making available immense computing power, it is far from being able to address the needs of solving complex optimization problems. The challenge is really in the algorithms front, and in recent years many meta-heuristic approaches such as evolutionary algorithms, simulated annealing, tabu search, ant colony optimization and neural networks have surfaced as viable stochastic optimization techniques. It has also become a norm in algorithm development to “mix-and-match” between the various available techniques to develop algorithms that are able to capitalize on the strengths and merits of the various techniques.

However, the increasing complexity of both problems and algorithms makes algorithm development and problem-solving more and more challenging. Conventional algorithm development methodology usually requires significant effort in code generation and modifications. Furthermore, the quality of the algorithm depends very much on the designer’s experience, level of knowledge

specific to the problem being addressed, as well as the algorithm flow structure. A programmer without profound algorithm design experience will find developing an effective and robust algorithm extremely challenging. Not to mention that in practical scenarios, users may face with various requirements of the algorithm. An algorithm that solves one particular class of problems well may not work for other classes of problems or scenarios at all. This makes the algorithm development for complex real-life applications even tougher.

With these justifications, a paradigm of algorithm development and implementation through a problem-solving environment development platform ADEP was proposed in order to address the need for rapid generation of efficient algorithms for solving a targeted class of real-life problems. ADEP is an acronym for algorithm development environment for permutation-based problems. Within it is a framework of novel hierarchical tree structure, which is termed as left variation – right property (LVRP) tree structure, for managing the procedure module library. Based on this novel LVRP management and object-oriented design concept, users can configure and generate a meta-heuristic algorithm via an intuitive graphical user interface. More importantly, ADEP through its learning engine is able to automatically optimize the configurations of the algorithm based on user-specified requirements. This way, both the efficiency and effectiveness of algorithm development are enhanced significantly.

The usefulness and effectiveness was validated on some real-life applications as well as some benchmark problems. It is shown that ADEP is able to reduce the algorithm development time significantly. More importantly, ADEP is able to automatically produce algorithms based on users' specified requirements and produce human-competitive or even better performance in a class of well-known benchmark problems.

Contents

ACKNOWLEDGEMENT	III
SUMMARY	IV
CONTENTS	VII
LIST OF FIGURES	XI
LIST OF TABLES	XIII
CHAPTER 1 INTRODUCTION	1
1.1 INTRODUCTION	1
1.2 MOTIVATIONS	2
1.3 OBJECTIVES	3
1.4 CONTRIBUTIONS	4
1.5 THESIS OUTLINE	6
CHAPTER 2 LITERATURE REVIEW	8
2.1 INTRODUCTION	8
2.2 PSES - A SURVEY	10
2.2.1 <i>MATLAB</i>	<i>10</i>
2.2.2 <i>EDSS</i>	<i>11</i>
2.2.3 <i>ELLPACK</i>	<i>11</i>
2.2.4 <i>HiQ</i>	<i>12</i>
2.3 PSES OR TOOLS FOR ALGORITHMS DESIGN	12
2.3.1 <i>HEEDS</i>	<i>13</i>
2.3.2 <i>Graphical environment for managing and developing applications</i>	<i>14</i>

2.3.3	<i>TOMLAB</i>	14
2.3.4	<i>EASEA and EAML</i>	15
2.3.5	<i>ParadisEO</i>	16
2.3.6	<i>EasyLocal++</i>	16
2.4	METHODS OR TOOLS FOR OPTIMIZING META-HEURISTICS	18
2.4.1	<i>Racing Algorithms for Metaheuristics Tuning</i>	19
2.4.2	<i>Evolving Evolutionary Algorithms Using Linear Genetic Programming</i>	21
2.4.3	<i>NeuroEvolving Robotic Operatives (NERO)</i>	22
2.4.4	<i>Genetic Programming</i>	22
CHAPTER 3	META-HEURISTIC ALGORITHMS OVERVIEW	24
3.1	GENETIC ALGORITHM (GA)	24
3.2	ANT COLONY OPTIMIZATION (ACO)	29
3.3	VARIOUS LOCAL SEARCH PROCEDURES	31
CHAPTER 4	ALGORITHM DEVELOPMENT ENVIRONMENT FOR PERMUTATION-BASED PROBLEMS	34
4.1	SYSTEM OVERVIEW	34
4.2	GENETIC ALGORITHM MODULE.....	37
4.3	ACO MODULE.....	51
CHAPTER 5	ALGORITHMS CONFIGURATION AND GENERATION.....	56
5.1	CODE GENERATION	59
5.2	CUSTOMIZING GENETIC ALGORITHMS	61
5.3	CUSTOMIZING ACO.....	63
5.4	CUSTOMIZING LOCAL SEARCH	64
CHAPTER 6	CASE STUDIES	66
6.1	<i>N-QUEEN PROBLEM</i>	67
6.1.1	<i>Problem Description</i>	67

6.1.2	Objective and fitness function.....	68
6.1.3	Algorithm Configuration, Generation and Customization.....	69
6.1.4	Simulations.....	71
6.2	SITE VISITATION SEQUENCING PROBLEM FOR A SINGLE UNMANNED RECONNAISSANCE AERIAL VEHICLE (URAV)	73
6.2.1	Problem Description.....	73
6.2.2	Objective and Fitness Function.....	74
6.2.3	Algorithm Configuration, Generation and Customization.....	74
6.2.4	Simulations.....	75
6.3	VEHICLE ROUTING PROBLEM (VRP)	76
6.3.1	Problem description.....	76
6.3.2	Solution Representation	79
6.3.2.1	Genetic representation 1	79
6.3.2.2	Genetic representation 2	82
6.3.3	Objective Function and Fitness Function	84
6.3.4	Local Search for TSP	87
6.3.5	Algorithm Configuration, Generation and Customization.....	88
6.3.6	Simulations.....	88
6.3.6.1	Comparison on CVRP benchmark instances	89
6.3.6.2	Comparison on FTDS instances	90
6.3.6.3	Comparison with other algorithms	94
6.3.7	Performance Analysis	95
6.4	SUMMARY OF THE CASE STUDIES	98
CHAPTER 7	AUTOMATIC CONFIGURATION ENGINE	100
7.1	MOTIVATION.....	100
7.2	AUTOMATIC CONFIGURATION ENGINE	101
CHAPTER 8	VALIDATION ON AUTOMATIC CONFIGURATION ENGINE.....	108

8.1	QUADRATIC ASSIGNMENT PROBLEM.....	108
8.2	AUTOMATIC CONFIGURATION ENGINE SETUP FOR QAP	113
8.2.1	<i>Algorithm evaluation</i>	113
8.2.2	<i>Trace update and initialization</i>	116
8.3	SIMULATION RESULTS AND ANALYSIS	117
8.3.1	<i>Simulation results</i>	118
8.3.2	<i>Trace analysis</i>	122
CHAPTER 9	CONCLUSIONS AND FUTURE DIRECTIONS.....	127
9.1	CONCLUSIONS	127
9.2	FUTURE RESEARCH	129
9.2.1	<i>Short term</i>	129
9.2.2	<i>Long term</i>	130
CHAPTER 10	AUTHOR'S PUBLICATIONS.....	132
BIBLIOGRAPHY		135
APPENDIX		157

List of Figures

Figure 3-1. General Template of Genetic Algorithm	26
Figure 4-1. ADEP architecture	34
Figure 4-2. Possible GA Flowcharts	39
Figure 4-3. Offspring Producer	42
Figure 4-4. Individual Operation.....	44
Figure 4-5. Population Initialization	45
Figure 4-6. Population Update	47
Figure 4-7. Population Evaluation	49
Figure 4-8. Termination Condition	50
Figure 4-9. Possible ACO Flowcharts.....	52
Figure 4-10. Construction of Ants.....	54
Figure 4-11. Pheromone Initialization	55
Figure 4-12. Pheromone Update	55
Figure 5-1. LVRP Traversal Rule	57
Figure 5-2. Configuring Offspring Procedure by Traversing the Tree	57
Figure 5-3. Components of GA.....	61
Figure 6-1. Hybrid Genetic Algorithm (adapted from [C3])	70
Figure 6-2. Illustration of a 3-Vehicle Routing Problem	77
Figure 6-3. Genetic Representation 1	81
Figure 6-4. Genetic Representation 2.....	83
Figure 6-5. A sample routing plan on Singapore Island	94

Figure 6-6. Virtual Trucks Using GR2.....	97
Figure 7-1. Automatic Configuration Engine - Training Phase	104
Figure 7-2. Trace Initialization.....	105
Figure 7-3. Trace Update	107
Figure 8-1. Resulting hybrid genetic algorithms.....	118
Figure 8-2. Trace Analysis - Population Initialization	124
Figure 8-3. Trace Analysis – Population Size.....	125

List of Tables

Table 6-1. <i>N</i> -Queen Problem Solution Density	68
Table 6-2. Simulation Results on <i>N</i> -Queen Problem	72
Table 6-3. Simulation results for site visitation sequencing problem	75
Table 6-4. Comparison on a 50-station benchmark problem	89
Table 6-5. Comparison on a 75-station benchmark problem	90
Table 6-6. Distance matrix in FTDS	92
Table 6-7. Comparison on FTDS Instance	93
Table 6-8. Simulations on benchmark problems.....	95
Table 8-1. Simulation results on QAP benchmark problems.....	119
Table 8-2. Comparison with two other algorithms for QAP.....	121

Chapter 1 Introduction

1.1 Introduction

Combinatorial optimization problems arise in a large number of real-life applications. Examples include the quadratic assignment problem (QAP) [1] and the traveling salesman problem (TSP) [2]. On the other hand, the complexity of these problems increases with problem dimension exponentially, a characteristic feature of NP-hardness. This NP-hardness renders these problems to be computationally intractable. Exact methods which are designed to guarantee optimal solutions, become unmanageable as the size of the problems increases. As a result, many heuristics based approaches have surfaced to address the needs to search in relatively short time for high-quality solutions for these problems. Modern heuristics based approaches are generally classified as meta-heuristic algorithms or simply meta-heuristics. Meta-heuristics are essentially high-level search strategies for exploring solution spaces using different methods. They are approximate search algorithms and are usually non-deterministic. Examples of meta-heuristics include evolutionary computation, simulated annealing, tabu search, neural networks, adaptive greedy approach, and more recently swarm intelligence approaches such as ant colony optimization.

1.2 Motivations

In practice, some efforts in customizing the meta-heuristic algorithms are required to suit the specific nature of the problem domain. It is believed that more meta-heuristic algorithms will be devised in future in order to address the need for efficient search algorithms to tackle real-world complex problems. Furthermore, the hybridization of these meta-heuristics will offer another layer of abstraction for higher level heuristics in solving complex optimization problems.

However, conventional algorithm development requires a great deal of low-level code generation and modification. It is tedious and inefficient, and hence not able to keep up with the explosive increase in problem complexity.

More importantly, the performance of the designed algorithm depends very much on the designer's experience on algorithm design and the level of knowledge specific to the domain of the problem being addressed. One without profound algorithm development experience will find it challenging to develop a powerful algorithm for the problems being addressed. Unfortunately so far, there have been few attempts to address the need for automatic optimization of the configurations of meta-heuristic algorithms.

With these considerations in mind, there is a need for a more efficient and effective algorithm development methodology. A paradigm for novel algorithm development through an integrated problem-solving environment is introduced. It serves as a platform for designing and generating meta-heuristic algorithms for solving a class of combinatorial optimization problems.

1.3 Objectives

This thesis outlines the novel algorithm development methodology implemented through an integrated problem-solving environment for solving a class of combinatorial optimization problems.

It is implemented as a generic software platform with an open and friendly interface for users to configure, explore and generate algorithms for solving various combinatorial optimization problems. Furthermore, the system should be able to assist the user in choosing, configuring, generating and more importantly, optimizing a meta-heuristic algorithm for solving the domain-specific problem being addressed. The system, which is referred to as Algorithm Development Environment for solving Permutation-based optimization problem (ADEP) can be extended to handle optimization problems with other, not necessarily permutation-based solution representation.

1.4 Contributions

By now, the objectives of the project are well achieved. The proposed algorithm development methodology has been realized and tested together with the completion of the problem-solving environment – ADEP. In particular, it focus on optimization problems with permutation-based solution structures. The contributions of this thesis are summarized below.

ADEP facilitates the configuration and generation of meta-heuristics based solution methods for real-life applications of combinatorial optimization problems. The novel features of ADEP are as follows:

- 1) ADEP adopts a flowchart as the representation of an algorithm and implements a user-friendly graphical interface to facilitate the configuration of the algorithm for the user.
- 2) ADEP incorporates a procedural library, which contains various heuristic routines for users to explore numerous possibilities of the meta-heuristics configurations.
- 3) The system uses a novel data structure, namely left variation – right property (LVRP) tree, to manage the various procedures, routines, properties in the procedure library. The LVRP tree is scalable and portable, open to incorporate new procedural elements.

- 4) A code generator was developed based on the LVRP tree structure, which adopts the Objective Oriented (OO) design concept. With this feature, code generation and modification is much less tedious than that of conventional algorithm development.
- 5) The usefulness of the proposed algorithm development methodology implemented in ADEP was verified by studying several test cases of combinatorial optimization problems including a real-world application. The development of the problem-solving algorithms using ADEP was achieved in a significantly reduced time frame compared with conventional algorithm development.

Furthermore, a novel evolutionary learning methodology based on the LVRP tree structure was developed to automatically optimize the configurations of an algorithm, taking the domain specific features of the problem and the particular user requirements into account for solving the problem being addressed.

- 1) The LVRP tree structure enables users to map the configuration process to a tree traversal problem. An evolutionary learning methodology is proposed and implemented to search for all the traversal possibilities (i.e. possible configurations). It is able to learn from the domain specific problem being addressed with considerations of the users' particular requirements to uncover the best set of configurations of the algorithm.

- 2) The proposed learning methodology was tested on a well-known combinatorial optimization problem, namely quadratic assignment problem (QAP). By training based on sample problem instances, the learning engine was able to derive a traversal path in the LVRP tree. The traversal path represents a solution methodology that is efficient for the problem instances being addressed. As a result, an algorithm derived through training achieves human competitive performance level or even better than most reported algorithms in literatures.

It is worth noting that although ADEP focuses on the class of combinatorial optimization problems with permutation-based solution structure, the ideas and methodologies used in ADEP can be easily extended to address other problems with different string-based solution structures. This feature can serve as a useful extension of my current work.

1.5 Thesis Outline

The thesis is organized in the following manner.

Chapter 2 focuses on the literature review of problem-solving environment technology and existing means of algorithm development for solving various problems.

Chapter 3 reviews and summarizes the various classes of meta-heuristic algorithms for solving combinatorial optimization problems.

In Chapter 4, the proposed environment Algorithm Development Environment for Permutation-based optimization problem (ADEP) is presented.

Chapter 5 outlines the approach of configuring and generating meta-heuristic algorithms using ADEP.

Chapter 6 presents several case studies of using ADEP to configure and generate meta-heuristic algorithms for solving some combinatorial optimization applications.

Chapter 7 presents a novel methodology proposed within the framework of ADEP to automatically optimize the configurations of the meta-heuristic algorithms.

Chapter 8 gives the evaluation of the proposed methodology. A case study on a well-known combinatorial optimization problem is presented.

Finally, some conclusions and directions on future work are presented in Chapter 9.

Chapter 2 Literature Review

2.1 Introduction

The concept of problem solving environment (PSE) was first proposed in the early 1960's [3-5], but concrete research only began in the 1990's. The term "problem solving environment" (PSE) means different things to different people because it is relatively novel and development has started only very recently. A general definition of PSE in [6] is as follows:

A problem solving environment is a computer system that provides all the necessary computational facilities to solve a target class of problems.

These facilities include advanced solution methods, automatic and semi-automatic selection of solution methods, and ways to easily incorporate novel solution methods. Moreover, a typical characteristic of PSEs is that they use "language" which is natural for the target class of problems. This way, users can run them without specialized knowledge of the problem-domain being addressed and the underlying computer hardware or software.

The goal of PSE technology is to exploit the existing enormous hardware and/or algorithm power to deliver “cheap and fast” solutions. PSEs are one important component in the effort to remedy the lack of increase in programming power for science and engineering.

The definition of PSE can be summarized by the following abstract equation [6]:

$$PSE = \textit{Natural Language} + \textit{Solvers} + \textit{Intelligence} + \textit{Software Bus} \quad (1)$$

where the software bus represents the computing infrastructure (machines, systems, networks etc.) tying everything together. The natural language is used by the system to “understand” the user rather than the other way around. For example, many PSEs, which were introduced in the 1990s, use the “language” of the respective field and often employ modern graphical user interfaces to make the software easy to use for users in fields other than computer science. Normally, mathematical and scientific notations are used with text, graphical or geometric tools. The PSEs must have a collection of solvers as there will not be one single solver that is best for the problems of interest. The PSEs also should have sufficient intelligence and knowledge about the target problems, and in turn, to decide how to organize the solvers used and how to make use of the underlying computing services.

There are many PSEs developed for various problem-domains in the last decades. Some of them are summarized in the next section.

2.2 PSEs - A Survey

I survey some of the existing PSEs that are available commercially or as public domain packages. The survey is not meant to be exhaustive but rather to provide a general overview of the different features and capabilities of typical PSEs.

2.2.1 MATLAB¹

MATLAB is a PSE from Mathworks based on a programming language for matrix computations. Its origin can be traced back to the early 1970s when it was purely an interactive matrix/vector language. It has evolved over the years with the incorporation of good graphics and toolboxes to broaden the problem domains it can address. It is available on most computing platforms and the suite of toolboxes include control systems, databases, finance, fuzzy logic, image processing, optimization, partial differential equations, statistics, symbolic mathematics, etc. Therefore MATLAB is probably the most widely used PSE for computational problem-solving. It provides interfaces for a number of generic facilities (web browsers, and Fortran, ActiveX) and in the Windows environment, uses the word processor to provide a notebook facility.

¹ <http://www.mathworks.com/products/matlab>

2.2.2 EDSS²

The Environmental Decision Support System (EDSS) is a PSE that provides an advanced modeling and analysis system for environmental scientists, engineers, policy makers, and educators. EDSS's design goals include allowing modelers to incorporate new knowledge with minimal effort, providing flexibility to model diverse issues. It is targeted at helping decision makers reliably generate and understand more information with less effort, and contributing to a community modeling and analysis system. To achieve these goals, EDSS includes a paradigm, components, and tools for building air quality models from interchangeable components, data analysis and manipulation tools, an extremely fast emissions processing package, a graphical computation manager and an experimental optimization-based strategy development tool.

2.2.3 ELLPACK³

ELLPACK is a very high level, portable PSE for partial differential equations (PDEs) based applications. It adopts an extension of Fortran as the natural “language”. It greatly reduces coding for most of the computation, but still allows one to do special processing. The methods written by users can be named and reused following the ELLPACK framework for further problem-solving.

² <http://envpro.ncsc.org/EDSS>

³ <http://www.cs.purdue.edu/ellpack/>

ELLPACK incorporates over 50 problem solving modules, including some of the best and most current software in the world. A knowledgeable user can tailor ELLPACK by adding new problem solving or analysis modules.

2.2.4 HiQ⁴

HiQ is a PSE which helps analyze, visualize and document the solution of science and engineering problems. It is organized as a notebook that is created as a problem is solved, sophisticated 2D and 3D displays are created in the notebook and can be modified directly. The analysis is made using the HiQ-Script language, by direct access to MATLAB (see Subsection 2.2.1) or by access to user created objects which encapsulate existing software. HiQ-Script include a wide array of built-in mathematical, numerical, statistical, visualization and utility functions. HiQ is integrated into the Windows operating system so generic facilities like Word, Excel and Power Point are available to the notebook environment.

2.3 PSEs or Tools for Algorithms Design

Nowadays, there are many real-world applications and projects which require optimization. As the size and complexity of the optimization problems increase, it becomes more and more difficult to solve these problems quickly and accurately. Therefore in the recent decade, a particular class of PSEs or tools, which are

⁴ <http://www.natinst.com/hiq/>

specifically developed for solving various optimization problems in real-world applications, has drawn the attention of many researchers. My work belongs to this category and I briefly introduce examples of such PSEs in this subsection. Their difference from ADEP will be analyzed and presented.

2.3.1 HEEDS⁵

HEEDS stands for *Hierarchical Evolutionary Engineering Design System*. It is a PSE incorporating various optimization techniques, including genetic algorithm (GA). It provides user-friendly interface for exploring various optimization tools for solving a myriad of optimization problems being addressed. This class of platforms [7-9] facilitates the process of algorithm design and thus enhances the efficiency of problem-solving greatly. However, they are essentially simulation environments. Though various algorithms can be configured and executed efficiently in these environments, the execution depends on the entire system. For many applications which require embedded real-time solver, this class of environments does not offer the flexibility to configure an efficient stand-alone program, *albeit* a turnkey problem-solving algorithm.

⁵ <http://www.redcedartech.com/ot/>

2.3.2 Graphical environment for managing and developing applications⁶

In [10], a graphical environment to assist users in generating stand-alone application rapidly, is presented. It contains a modules library managed by a spreadsheet program, and adopts other software reuse technology. Users are able to retrieve the procedures from the library with the friendly GUI it provides. Based on the procedures selected, a stand-alone application is generated automatically. However, this invention only addresses the programming issue. It does not involve any algorithm design knowledge. Therefore it may not meet the needs of solution methodology whereby complex algorithms like meta-heuristics need to be developed.

2.3.3 TOMLAB⁷

TOMLAB is a general optimization environment in MATLAB. It is essentially a MATLAB toolbox, which encapsulates various optimization procedures, including meta-heuristics such as genetic algorithms. It supposedly makes it easier for users to retrieve the integrated procedures and routines to build up an application in the form of source codes. However, it cannot generate the source codes automatically and the user has to deal with the functional programming language, i.e. MATLAB programming language. This is a big obstacle for users who may not be familiar

⁶ U.S. Pat. No. 6,286,017

⁷ <http://tomlab.biz/>

with MATLAB language. Furthermore, it requires the support of an expensive third party software environment, i.e. MATLAB.

2.3.4 EASEA and EAML

Easy Specification of Evolutionary Algorithm (EASEA) [11] is a high-level script language dedicated to the specification of evolutionary algorithm (EA). It retrieves various EA procedures and routines from well-known EA libraries: GALib⁸ and EO⁹. Users compose the *.ez* file in EASEA language for configuring an evolutionary algorithm. The *.ez* file is then compiled by EASEA compiler to C++ files. The generated C++ files are in turn compiled and linked with the corresponding library to produce an executable file implementing the evolutionary algorithm specified in the original *.ez* file. This way, the most tedious codes generation part of algorithm development may be avoided. This way, the efficiency of application development may be enhanced significantly. However EASEA, as well as another similar language Evolutionary Algorithm Modeling Language (EAML) [12], is still a form of text-based language, although it is of higher level than any functional programming language. These two script languages both require a specifically designed compiler to translate the script language to a functional programming language. Furthermore, it is necessary for general users to be familiar with the syntax of the language.

⁸ <http://lancet.mit.edu/ga/>

⁹ <http://eodev.sourceforge.net/>

2.3.5 ParadisEO¹⁰

ParadisEO is a global framework which is an amalgamation of several existing packages. It is a templates-based, ANSI-C++ compliant metaheuristics computation library based on EO¹¹ library.

ParadisEO incorporates the packages for a wide range of metaheuristic algorithms. Examples include *population-based metaheuristic* (genetic algorithm etc.), *single solution metaheuristic* (local search), *multi-objective metaheuristics* and *parallel and distributed metaheuristics* and so on.

However, ParadisEO is essentially a library. Compared with ADEP, though ParadisEO has a much larger procedural library, users still need to deal with a considerable amount of coding to incorporate the libraries to their own applications. Moreover, there is no friendly GUI or interactive guidance to facilitate the user to configure a feasible metaheuristic algorithm.

2.3.6 EasyLocal++¹²

EasyLocal++ is an object-oriented software framework for developing various local search algorithms in C++. The basic idea of it is to capture the essential features of most local search algorithms, which is similar to the idea of code

¹⁰ <http://paradiseo.gforge.inria.fr/index.php?n=Paradiseo.Home?from=Main.HomePage>

¹¹ <http://eodev.sourceforge.net/>

¹² <http://tabu.diegm.uniud.it/EasyLocal/>

generation in ADEP. These features are managed in a four-layer structure: *solvers*, *runners*, *helpers* and *basic data classes*. *Solvers* represent the external software layer of EasyLocal++; the *runners* represent the algorithmic code of the framework; *helpers* represent the local search problem and finally basic features like the problem representation belong to the lowest level *basic data classes*. Each layer has its abstract functions. To use these features, the user needs to define and implement those functions. EasyLocal++ allows expert-user to customize the framework to achieve various and even new local search algorithms.

However, compared with ADEP presented in this thesis, EasyLocal focuses narrowly on the local search algorithms. While ADEP aims to help develop various metaheuristics like genetic algorithm (GA), ant colony optimization (ACO) and various local search procedures etc.

Furthermore, EasyLocal++ is essentially a software library. To use EasyLocal++, users still “have to define the data classes and the derived helper classes”¹⁰. That means the user still need to do a considerable amount of coding. On the contrary, one of the objectives of ADEP is to free the user from low-level coding task as possible. The code generator reduces the user’s coding workload to the minimal level.

Finally, EasyLocal++ does not provide either text-based language interface or graphic user interface to facilitate the configuration of local search algorithms. In contrast, ADEP provides a friendly GUI as well as interactive guidance for the user to configure a feasible algorithm.

It is worth noting that there are many other software libraries like EasyLocal++ and ParadisEo. More examples can be found in [13]. However, they are essentially software libraries. The effort of incorporating the libraries into the user's work is still considerable compared with that of ADEP. Not to mention that none of them incorporates an algorithm optimizer like the automatic configuration engine in ADEP.

2.4 Methods or Tools for Optimizing Meta-heuristics

In recent years, researchers began to pay more attention developing technologies to automate the algorithm design process to reduce the reliance on human's experience and inference. Most of these studies are still in the infancy stage and there are even fewer attempts to address the needs for the optimization of various meta-heuristic algorithms. Their difference from ADEP will be analyzed and presented.

2.4.1 Racing Algorithms for Metaheuristics Tuning

In [14], a racing algorithm was proposed for configuring the parameters of meta-heuristics. It empirically evaluates a set of candidate configurations of parameters by discarding bad ones as soon as statistically sufficient evidence is gathered against them. This procedure was tested as an example on the configuration of an ant colony optimization (ACO) algorithm applied to the traveling salesman problem. Though it configures the ACO efficiently and the resulting ACO algorithm is competitive, this proposed racing algorithm is only able to tune the real number parameters in the meta-heuristics and it fails to address other configuration issues, like structures, procedure selections and so on. It is worth noting that another similar system exists, namely CALIBRA [15], which adopts fractional experiments strategy to tune the real number parameters.

As an extension of the racing algorithm above, a series of algorithms, namely F-race algorithms, for tuning metaheuristics were presented in Birattari's Ph.D. thesis [16]. The basic idea of the F-race algorithm is to formulate the tuning problem as a search problem in a multi-dimensional space. Each dimension represents a real number parameter, choices of structure or choices of procedures, etc. Each tuning engine borrows the idea from racing algorithm. It works by evaluating the candidate configurations and meanwhile eliminating the inferior candidate configuration instances or branches based on a statistical test. This way, the search space is narrowed down significantly along with the progress of the tuning.

The experiments on several metaheuristics showed that the F-race algorithm can efficiently produce the metaheuristics configurations for solving well-known combinatorial optimizations like traveling salesman problem (TSP) and quadratic assignment problem (QAP) with impressive performance.

However, though Birattari provided the general framework of the F-race algorithm and applied it to several metaheuristics, the formulation and implementation of the tuning algorithm may not be easy for a particular application. To use the F-race algorithm, the user has to abstract the “features” to be tuned from a metaheuristic algorithm and formulate them into a multi-dimensional space. The F-race algorithm narrows down the search space by eliminating some configurations or branches of the configurations. Yet, there is still a lower bound for the search complexity. When the search space is highly dimensional, the complexity of the search space could be impractically large for F-race algorithm. That is why in Birattari’s thesis, the F-race algorithm only applied to applications with very limited dimensions.

In contrast, the automatic configuration engine in ADEP works in a very different manner. It formulates the search space using a novel tree structure. Instead of eliminating candidate configurations along with the progress of tuning, ADEP adopts a self-adaptive learning mechanism to converge the search to a very good

configuration. This way, the current version of ADEP can handle tuning applications with more than 100 dimensions.

2.4.2 Evolving Evolutionary Algorithms Using Linear Genetic Programming

In [17], a linear genetic programming approach was proposed to evolve the evolutionary algorithms. The basic idea is to represent a candidate evolutionary algorithm (EA) as a variable-length sequence of statements in C language. Subsequently, the linear genetic programming technique is used to evolve the sequence of C statements.

Compared with ADEP, this approach is strongly constrained to only standard evolutionary operations like crossover, mutation etc. It does not address and capitalize on the multitude of metaheuristics available. Also, the encoding the EA into a sequence of C code is not an easy task. Usually, specialized knowledge and experience may be required.

Finally, the authors evolved an EA for solving quadratic assignment problem (QAP). The simulation results show that it is no match to the evolved hybrid genetic algorithm by ADEP, which will be presented in Chapters 7 and 8.

2.4.3 NeuroEvolving Robotic Operatives (NERO)

In [18], an evolutionary system, namely NeuroEvolving Robotic Operatives (NERO), was proposed. It encodes the neural networks as artificial chromosomes and uses genetic algorithm to evolve the chromosomes (i.e. neural networks) for the purpose of a real-time game. It optimizes not only the parameter settings, but also the topological structures of the neural networks. It is also able to adjust the neural networks according to dynamic changes of the game scenarios in real time. NERO is able to achieve competitive results and addresses the needs of quick generation of complex systems (i.e., neural networks in this case) with excellent performance. However, NERO is dedicated to neuro-evolution. It does not address the issue of evolving other general meta-heuristic algorithms as ADEP does.

2.4.4 Genetic Programming

Koza [19, 20] proposed a novel problem-solving technology, namely genetic programming. It encodes a solution method or system (in term of a computer program) into a tree based artificial chromosome and then adopts the operations in genetic algorithm to evolve the chromosomes (i.e. the program). Genetic programming not only optimizes the parameter settings, but also the structures of the solution method for a given problem. Genetic programming successfully produced many human-competitive results in a wide range of real-life applications.

However so far, it proves only applicable to relatively simple systems, like function approximation, RLC circuits design or even a programming problem as in [21]. It still cannot address the need of automatic generation of complex meta-heuristic algorithm for general optimization as required in real-life applications.

Chapter 3 Meta-Heuristic Algorithms Overview

My work focuses on a computational platform for algorithm development to address the needs of problem-solving involving a class of combinatorial optimization problems whereby the solution structure can be coded as a permutation string. In particular, the focus of my work is the development of a class of algorithms, which is referred to as meta-heuristics. Meta-heuristic algorithms are solution methods that orchestrate an interaction among various heuristic search procedures so as to draw on the merits of each heuristic approach. They have proven to be remarkably effective in solving complex optimization problems in reasonable time. In this chapter, a brief survey on some well-known metaheuristics is presented.

3.1 Genetic Algorithm (GA)

Genetic algorithm (GA) was first proposed by John Holland [22] in 1975. It is a class of adaptive search methods to uncover approximate solutions to search problems. It is a branch of evolutionary algorithm (EA) which is supposed to be more general and encompass the various evolutionary based techniques such as evolutionary strategy, evolutionary programming and genetic programming.

Genetic algorithm (GA) is a class of search techniques used in computer science to find approximate solutions to optimization and search problems. GA uses techniques inspired by evolutionary biology such as inheritance, mutation, natural selection, and recombination (or crossover). In GA, candidate solutions to a given problem are encoded in abstract representations. Although it is common for solutions to be represented as binary strings in GA, different encodings are also possible to suit different problem domain. A general template of GA is presented in Figure 3-1. The evolution process starts with a population of initial individuals and goes through cycles of progressive modifications. Each cycle is referred to as a generation. In each generation, individuals to be designated as parents are first selected. In most cases, fitter individuals are given higher chances of being selected as parents. Through a series of artificial genetic operations, like crossover and mutation (usually with some probability), the parent individuals reproduce. This way, partial “characteristics” of each parent are inherited by the offspring at the genetic level. This reproduction process is repeated until sufficient offspring are created. Finally, the offspring, together with the parent individuals, form a new population for the subsequent evolutionary process in the next generation. In most cases, fitter individuals are awarded with higher chances of survival. This way, “desirable” characters of both parents and offspring are preserved. The above steps are repeated until the termination condition is met and the best solution ever uncovered will be reported as the result of the GA search.

```
population initialization;  
while termination condition is false  
    repeat  
        parents selection;  
        crossover;  
        mutation;  
        evaluate offspring;  
    until sufficient offspring created  
population update;  
endwhile
```

Figure 3-1. General Template of Genetic Algorithm

Due to its robustness, generality and adaptability, GA has been developed and applied to a wide range of applications since its inception in the 1970's. Research since then has focused on almost every aspects of GA.

It has been empirically and qualitatively proved that high diversity of the initial population is usually preferred for uncovering good quality solutions [23, 24]. To obtain high diversity of the population at the beginning of GA, random initialization of the population of candidate solutions is usually adopted. However, some theoretical research [25, 26] shows that in general, randomly chosen solutions do not necessarily cover the search space uniformly. Some statistical methods of initialization may benefit the convergence of the GA. The issue on the size of the population has been approached from several theoretical points of view.

However, the underlying idea has always been a trade-off between efficiency and effectiveness. It is not difficult to understand that in general, too small a population does not allow sufficient room for exploring the search space. It converges quickly but usually yields low quality solutions. On the other hand, too large a population size gains sufficient exploration in the search space, but no solution can be expected in a reasonable time. Glodberg [27, 28] had made an attempt to address this issue. He claimed that the proper population size increases as an exponential function of the problem dimension. However, some experimental results [29, 30] show that this relationship does not stand. There are several attempts to define the minimum setting of the population size, For example, Reeves suggested in [31] that a population size of $O(\log N)$, where N is the problem dimension, should be sufficient to cover the search space. However, none of these conclusions has been proven to be universally viable.

Parents selection is used to choose certain individuals from the existing population pool for the purpose of reproduction. The idea of most selection schemes is that fitter individuals are awarded more chances of being selected. A simple and well-known sampling scheme is roulette wheel selection (RWS) [24]. Other selection schemes include tournament selection, random selection, etc.. Although most of the GA applications still adopt RWS, empirical research on comparison of the selection schemes [32] has shown the superiority of stochastic universal selection [33] over RWS. In recent years, a new selection scheme, namely gender specific

selection [34-36], has received greater research attention. It classifies the population member into several gender groups and the crossover can only be carried out with parents from different gender groups. This way, the diversity of the population is preserved better than conventional GAs with single sex.

Population update is to determine the new population members based on the parent and offspring generation. De Jong [37] proposed a well-known concept of *elitism* and *population overlaps*. The idea is to preserve the best individuals in the parents and replaces the rest of the parents with the offspring. Many GA applications replace the worst members, such as steady-state reproduction. However, Goldberg and Deb [38] pointed out that this might introduce great selective pressure into the search and usually required large population size to avoid a rapid loss of diversity.

In the last decade, there has been a popular trend towards hybridizing GA with other heuristic procedures to enhance search performance [39-43]. The latest research leads to a new branch of GA, namely memetic algorithms [44]. Memetic algorithm mimics the communication and fusion of different social cultures. Each procedure component is regarded as a “meme”, which means “culture”. Memetic algorithm is also frequently called hybrid genetic algorithm (hybrid-GA), genetic local search (GLS) etc. [44].

3.2 Ant Colony Optimization (ACO)

Ant colony optimization algorithm (ACO), introduced by Marco Dorigo in 1992 [45], is a probabilistic technique for solving hard combinatorial optimization problems. It is inspired by the behavior of ants in finding paths from the colony to food. In the real world, ants (initially) wander randomly, and upon finding food return to their colony while laying down pheromone trails. If other ants find such a path, they are likely not to keep traveling at random, but to instead follow the trail, returning and reinforcing it by laying more pheromone if they eventually find food.

ACO algorithms have been used to produce near-optimal solutions to many combinatorial optimization problems such as traveling salesman problem (TSP) [46, 47], quadratic assignment problem (QAP) [48-50], vehicle routing problem (VRP) [51-54] and so on.

The recent improvement over ACO mainly focuses on two aspects. One aspect concerns the solution construction methodology. For example, much research has been dedicated to investigate the influence of the solution construction order – either a fixed order throughout the entire run [55] or a random order chosen based on uniform distribution [56]. In early development of ACO applications, such as the ant system (AS) in [55], domain specific information is often involved in the construction of the solution. However, later research and proposed ACO applications, such as fast ant system (FANT) [57], MAX-MIN Ant System

(MMAS) [56] and hybrid ant colony system (HAS) [49] removed the domain specific information out of the solution construction process and found to be even more efficient. An extension of AS, named approximate non-deterministic tree-search (ANTS) was proposed in [58]. The application of lower bounds on the completion cost of a partial solution in ANTS makes it a significant improvement over AS. Unlike most ACO applications that used population-based search scheme, FANT searches by single ant.

The other aspect involves the pheromone initialization and update schemes. Examples of breakthroughs in this aspect include the MMAS proposed by Stützle in 1999. To avoid the dominance of few pheromone traces and premature stagnation of the exploration, MMAS defines upper and lower bound values for the pheromone trace. Any pheromone trace that exceeds the upper bound will be limited to the value of upper bound, and vice versa. The pheromone traces are correspondingly initialized as the upper bound. Furthermore, pheromone traces are only allowed to be updated by either the iteration best solution or the global best solution ever uncovered. In contrast, AS allows every solution in the population to update the pheromone traces. As a result, MMAS produced significantly better solutions than its counterworker in many problems such as TSP [59] and QAP [50].

Other research trends on ACO include parameter tuning [60], hybridization of other meta-heuristic algorithm [61, 62] and parallelization of ACO [63].

3.3 Various Local Search Procedures

Tabu search is a local search technique, which was first proposed by Glover [64, 65]. Tabu search enhances the performance of a local search method by using memory structures. It memorizes the paths that it has traversed so that redundant search can be avoided. Tabu search uses a local or neighbourhood search procedure to iteratively move from a solution x to a solution x' in the neighbourhood of x , until some stopping criterion has been satisfied. The acceptance or rejection of the neighbourhood movement is determined through the use of special memory structures, which is called tabu list, and a set of criteria, namely tabu rules.

Robust tabu search was first introduced by Taillard in 1991 [66]. It may be the simplest tabu search available while it produced very good solution when applied to several classes of combinatorial optimization problems, such as QAP [67]. Robust tabu search adopts two categories of memory – short term memory and long term memory. The short term memory records the movements in the near history and thus defines the points that are forbidden within the very localized region. On the other hand, the long term memory records the representative points, which is used to define the local regions that the search has covered. The long term memory helps to define the forbidden regions that should not be searched again.

Reactive tabu search [68] is an improvement over robust tabu search. Its working principle is the same as that of robust tabu search. However, when the search returns to a point that is in the short term memory, reactive tabu search perturbs the search by randomly modifying the current solution instead of simply rejecting the movement. Furthermore, reactive tabu search applies the adaptive control in the search process. When no improvement is found for many iterations, it decreases the size of the short term memory to loose the tabu rule.

Stuetzle [41, 69] proposed new class of local search, namely iterated local search (ILS), with idea similar to reactive tabu search. ILS applies a certain form of local search procedure to exploit a local region of the entire search space. If the local search failed to find any improvement for many iterations, ILS regards it as a sign of being trapped at a local optimum, which has been uncovered. To escape from the local optimum, ILS perturbs the search process by making significant modification on the local optimum randomly and assigns it as the starting point of the local search procedure for the next round. These steps are repeated until the termination condition is met. Obviously, ILS alone does not involve use of the memory as tabu search does. However, combining the ILS concept with tabu search procedure leads to much better performance on some combinatorial optimization problems [J2].

Simulated annealing [70] is a local search algorithm, which mimics the annealing process in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. The heat causes the atoms to become unstuck from their initial positions (a local minimum of the internal energy) and wander randomly through states of higher energy; the slow cooling gives them more chances of finding configurations with lower internal energy than the initial one. Simulated annealing iteratively explores the neighbourhood solutions by localized movements. The acceptance or rejection of the neighbourhood movement is controlled by the “states energy”. The states energy are updated (increase or decrease) as the search progresses in order to escape from the local optima. The criteria for controlling the update of the energy are called cooling/heating strategy.

Chapter 4 Algorithm Development Environment for Permutation-based Problems

4.1 System Overview

An integrated PSE, namely, Algorithm Development Environment for Permutation-based problems (ADEP) was developed. It provides a computational platform for developers and researchers to configure and generate meta-heuristic algorithms that are efficient for solving real-life combinatorial optimization problems. In particular, it focuses on the class of combinatorial optimization problems whereby the solution structure is encoded as a permutation string. It will be explained later how the technique is generic enough so as to extend the scope of its applicability on any other string-based solution structure.

Figure 4-1. ADEP architecture

CHAPTER 4. Algorithm Development Environment for Permutation-based Problems

Figure 4-1 shows the overall architecture of the entire system and the flow of the problem-solving process in ADEP. First, the users access a GUI which facilitates the specification of configurations of a meta-heuristics algorithm for the problem being addressed. Instead of relying solely on manual configuration of algorithms through the GUI, ADEP also provides the option for user to carry out automatic configuration of algorithms. Based on the users' predefined requirements and expectations on the performance of the algorithm as well as the support of the procedure library, ADEP searches and uncovers suitable algorithm configurations which best satisfy the user. After that, the configured algorithm (represented in a tree structure, which will be introduced below) is sent to the code generator. All the procedures, routines, parameter settings etc. are loaded from the library and linked together in the code generator to generate the source codes of the algorithm as a form of stand alone program. Finally, only minimal customizations on the generated program are required to adjust the algorithm to solve various real-world applications, with the specific knowledge of the problem-domain being addressed. The resulting algorithm can be used as stand-alone or embedded as the problem-solving engine part of a more complex system.

A framework for representing an algorithm is required to design and specify a meta-heuristic algorithm. Algorithm designers usually use either pseudo codes or flowcharts. The examples of using pseudo codes or other high-level languages to

CHAPTER 4. Algorithm Development Environment for Permutation-based Problems

describe a meta-heuristic algorithm have been mentioned earlier in previous work such as EASEA [11] and EAML [12]. As mentioned in Chapter 2, the main drawback of this method is that it still uses text-based languages, though it is higher level than any functional programming language. These two script languages both require a specifically designed compiler to translate the script language to a functional programming language. Furthermore, it is necessary for general users to be familiar with the syntax of the language. Pseudo codes tend to be less intuitive and thus difficult for users who have no profound knowledge of the high-level language or meta-heuristic algorithm design. Therefore in this work, flowchart is chosen as a means of representing an algorithm.

ADEP incorporates several algorithm modules, which are essentially the flowcharts of various classes of algorithms (e.g. GA), for configuring solution methods for solving the class of optimization problems being addressed. In the next few sections, I will introduce the various algorithm modules in ADEP and how to organize and manage the various procedures, routines, parameters, etc., associated with the algorithm through a novel tree structure, namely, left variation – right property (LVRP) tree structure. Basically all the structures and implementations in current version of ADEP will be presented. Though the scale is no match to some well-known libraries, the point here is to introduce a novel and scalable way to manage and manipulate various procedures.

Users or system integrators with moderate background knowledge on algorithm design and programming skill may use ADEP to configure and generate a meta-heuristic algorithm in the environment. The result is a stand-alone program generated based on very high level user-specifications. Although users are still required to have the basic knowledge of the problem being addressed and thus be able to define an appropriate objective function, minimal coding effort was required to generate and customize the program. This way, the efficiency of algorithm development is enhanced significantly. The focus of this work is on the class of optimization problems with permutation solution structure, since the solution of many real-life problems can be coded as a permutation string.

ADEP's procedure library encapsulates a wide range of meta-heuristic routines. The LVRP tree structure is proposed and developed to store and manage these procedures systematically. In the following sections, I will introduce all the algorithm modules within the framework of ADEP.

4.2 Genetic Algorithm Module

Figure 4-2 shows some possible configurations of genetic algorithm (GA) or hybrid genetic algorithm (hybrid-GA) flowcharts. From the figure, there are up to six components in the GA modules, in particular, *Population Initialization*,

CHAPTER 4. Algorithm Development Environment for Permutation-based Problems

Population Evaluation, Termination Condition, Offspring Producer, Individual Operation and *Population Update*. First, the *Population Initialization* procedure produces an initial population of chromosomes. Then, in the generation loop, these chromosomes are evaluated based on the fitness function. In most cases, the statistical information on the population also needs to be obtained through the *Population Evaluation* procedural module. After that, the population of individuals is designated as the parent generation. Using the *Offspring Producer* procedures (e.g. crossover and/or cloning), the parents reproduce to create members of the offspring generation. Although not necessary in GA, *Individual Operations* (e.g. mutation and/or local search) are often applied to alter or improve the genetic composition of these existing members of the offspring generation. It is worth noting that in this figure, the dotted arrow indicates an optional flow of the genetic algorithm. In particular, it indicates the conventional genetic algorithm without the augment of *Individual Operation* (which may be mutation or local search). In contrast, the solid line indicates the flow of hybrid genetic algorithm which is essentially a conventional GA augmented by one or several *Individual Operations*. Finally, following the *Population Update* mechanism, the resulting offspring generation together with the parents generation, constitutes a new population for further reproduction. This process is repeated until the GA search converges or the termination condition is met.

CHAPTER 4. Algorithm Development Environment for Permutation-based Problems

Fig. 4-2a

Fig. 4-2b

Fig. 4-2c

Fig. 4-2d

Fig. 4-2e

Figure 4-2. Possible GA Flowcharts

CHAPTER 4. Algorithm Development Environment for Permutation-based Problems

Each component of the modules is associated to a set of properties and attributes providing users with options for tuning and testing the various configurations of an algorithm. All the properties and attributes are managed and organized in the LVRP tree structure.

Figure 4-3 to Figure 4-8 show the LVRP tree structures for these six components respectively. There are two categories of nodes in the tree. One is called “real” node and the other is called “virtual” node. Any “real” node represents a “substantial” feature/attribute (like *Crossover* and *Mutation* etc.) of the algorithm. It is denoted by a solid line rectangle, ellipse, hexagon or star. In particular, a solid line hexagon represents a real number parameter; a solid line rectangle, as a root node, represents a component in either GA or ACO module as shown in Figure 4-2 and Figure 4-9; a solid line ellipse represents a “substantial” feature/attribute (other than real number parameter) in this component. Finally in some special cases, an entire algorithm module (like ACO module) can be used as a “real” node of the current algorithm component, these nodes are denoted by solid line star.

Each “real” node has exactly two children. One is “variations” as left child and the other is “properties” as right child. These two nodes are called “virtual” nodes which are denoted by dotted line ellipse and rectangle respectively. They don’t represent any “substantial” feature/attribute of the algorithm. They are used to

CHAPTER 4. Algorithm Development Environment for Permutation-based Problems

organize the “real” nodes of an algorithm in a systematic way. The meanings of the shapes are summarized as follows.

- Solid line rectangle: component in an algorithm module as the root node.
- Solid line ellipse: substantial feature/attribute except real number parameter
- Solid line hexagon: real number parameter
- Solid line star: another algorithm module
- Dotted line rectangle: virtual node “property”
- Dotted line ellipse: virtual node “variations”

It is worth noting that either “variation” or “property” node can have arbitrary number of “real” children. However, if a “virtual” node (either “variation” or “property”) has no child (which mean no inherent variation version, sub-routine or parameter), the virtual node will be omitted accordingly.

Figure 4-3 denotes the tree structure for the *Offspring Producer* component. It is used to generate the offspring from the parent population for subsequent reproduction operations. The left variation node of the *Offspring Producer* in the form of a dashed line elliptical node serves as the root of the left subtree to denote the variations of its parent, i.e. *Offspring Producer*, (e.g. Crossover, Cloning, etc.)

CHAPTER 4. Algorithm Development Environment for Permutation-based Problems

while the dashed line rectangular block “properties” serves as the right property node is the root of the right subtree to indicate the common properties of all the variations of *Offspring Producer*, such as *Parent(s) Selection*. Similarly, subsequent subtrees are managed and organized in similar structure. For example, as one of the variation of *Offspring Producer*, *Crossover* contains variations like *Uniform Crossover*, *Order-1 Crossover*, *One-point Crossover* and so on, which are organized in its left subtree. Meanwhile, all these variations have one common feature, i.e., *Crossover Rate*, a real number parameter which is represented in a hexagonal block. Therefore it is denoted as a node in the right subtree of *Crossover*. It is worth to note that the dashed line blocks “variation” and “property”, also define the rules for configuring the left and right subtrees. For the “variation” subtree, the user only needs to choose one child (i.e. one variation) and configures the corresponding subtree of the child. For “property” subtree, all nodes branching out from it need to be configured. This rule is recursively applied as the system guides a user to configure a feasible algorithm.

Figure 4-3. Offspring Producer

CHAPTER 4. Algorithm Development Environment for Permutation-based Problems

It is not difficult to understand that this hierarchical tree structure is open to accommodate other novel procedures or ideas for embellishment. The new procedures can be located at the same level as the existing procedures as their siblings. For example, other well-known or novel crossover operators, like partially matched crossover (PMX), can be easily introduced to the system by locating them as the siblings of the existing Uniform Crossover, Order-1 Crossover and One-point Crossover. Their corresponding attributes and features are managed using the same tree structure as mentioned earlier. Therefore this structure allows for a system that is open and easily scalable.

CHAPTER 4. Algorithm Development Environment for Permutation-based Problems

Figure 4-4. Individual Operation

Figure 4-4 shows the tree structure for the *Individual Operation* component. It is the set of procedures that are applied to an existing individual for improving its quality or altering its structure. Two forms of *Individual Operation* are shown for illustration, 2-exchange local search and mutate. Several variations of local search like tabu search as well as two forms of mutations (scramble mutation and windowed mutation) were also incorporated.

Figure 4-5. Population Initialization

Figure 4-5 shows the tree structure for the *Population Initialization* component. A genetic algorithm starts with an initial population to be designated as the parents. The *Population Initialization* procedure is responsible to generate such initial population. In this component, population size, an important GA parameter, is specified. There are three initialization mechanisms available, Random Initialization, Greedy Randomized Adaptive Search Procedure (GRASP) and Ant

CHAPTER 4. Algorithm Development Environment for Permutation-based Problems

Colony Optimization (ACO). The PSE framework described is tested on a well-known class of combinatorial optimization problems, namely the Quadratic Assignment Problem (QAP). Referring to Figure 4-5, the GRASP procedure (item 300) implemented is the one described in Li et al. [71] for solving QAP. Item 200 represents an independent instance of ACO procedure, which will be elaborated in greater details later. This way, the present system provides users the option to hybridize GA with ACO or GRASP in an intuitive manner. The quality of the initial population of individuals can have significant impact on the performance of the entire genetic algorithm. Therefore it is common to apply heuristics based local search to improve the quality of the initial population. ADEP provides users the option to apply *Individual Operation* on the initial population of individuals. It is worth noting that, item 100 in Figure 4-5 is an independent instance of *Individual Operation*. The configuration of it can be different from that of the *Individual Operation* in the generation loop (if any). The structure of this subtree in Figure 4-5 is omitted, which has been presented in details by means of Figure 4-4.

Figure 4-6. Population Update

Figure 4-6 shows the tree structure of the block *Population Update*. It is used to determine which offspring should be accepted into the selection pool for further reproduction of the next generation. The three schemes are shown as subtrees and a brief explanation of each scheme follows:

- PU_1: Overlapping population replacement with a certain number of elites (# of elites).

CHAPTER 4. Algorithm Development Environment for Permutation-based Problems

- PU_2: When a pre-specified number (update interval) of offspring are produced, only the best chromosomes from both parents and offspring are accepted and added into the selection pool.
- PU_3: Accepts only the child that is better than the worst existing parent and different from members in the existing population.

The number of elites and the update interval are the properties and are set as real number parameters of PU_1 and PU_2 respectively.

Figure 4-7. Population Evaluation

Figure 4-7 shows the tree structure of the block *Population Evaluation*. It is used for fitness evaluation of the current parent population. Therefore the Fitness Function, which is used to evaluate the fitness of an individual, is encapsulated in this component as its property. Some pre-processings, like fitness scaling and sorting, are often applied to the parent population before reproduction starts. Four types of well-known scaling methods were implemented: linear scaling, sigma truncation, power law scaling and ranking. Some are associated with real number parameters, which are set as properties associated to the corresponding methods of scaling.

Figure 4-8. Termination Condition

Figure 4-8 shows the tree structure of the block *Termination Condition*. It is used to control the modes of terminating the genetic algorithm. Three criteria were incorporated; Time, Max number of generations and Convergence. The time criterion terminates the genetic algorithm when the user-specified computational time (CPU time in second) limit is reached, while the Max # of generations criterion terminates the algorithm when the number of iterations reaches the user-specified number of generations. The Convergence criterion detects if the evolution converges by counting the number of successive generations clasped with no improvement in solution quality. Once the number exceeds a user-specified value, the evolution process is deemed to have converged and thus, is terminated.

4.3 ACO Module

Figure 4-9 shows some possible configurations of flowcharts for another well-known class of algorithms - Ant Colony Optimization (ACO) algorithms. From the figure, there are five blocks in the ACO modules. First, the *Pheromone Initialization* procedure initializes the artificial pheromone trails. Then, in the evolutionary loop, a population (colony) of artificial ants is guided by the pheromone trails or other heuristics based schemes to seek and uncover solutions. Each artificial ant represents one possible solution. These solutions are evaluated based on the objective function. The construction and evaluation of the ants

(solutions) are carried out in the block labeled as *Construction of Ants*. Though not necessary in ACO, in most cases, some heuristics based local search is applied to the existing ants to further improve the solution quality through the block *Individual Operation*, as shown earlier in Figure 4-4. The configuration of this *Individual Operation* block in ACO module can be different from that of the GA module. After that, the pheromone trails are updated based on the current population (colony) of ants and sometimes the history of the evolution process. This update is carried out through the block *Pheromone Update*. The updated pheromone trails are supposed to guide the artificial ants by improving the chances of seeking and uncovering better solutions in the next iteration. This process is repeated until the termination condition is met.

Fig. 4-9A

Fig. 4-9B

Figure 4-9. Possible ACO Flowcharts

CHAPTER 4. Algorithm Development Environment for Permutation-based Problems

Figure 4-10 shows the structure of the *Construction of Ants* block. It is used to construct a population (colony) of solutions or artificial ants, based on the pheromone trails delivered previously and possibly other heuristics schemes. For permutation solution string structure of length N , the pheromone trails can be represented as τ_{ij} ($1 \leq i \leq N, 1 \leq j \leq N$), which can be interpreted in two different ways. τ_{ij} may indicate the preference of allocating integer “ i ” to the j -th location in the permutation string. I refer to this “allocation” process as the pheromone model. This pheromone model is usually used for the class of assignment or allocation problems, like quadratic assignment problem (QAP). τ_{ij} can also be interpreted as the preference of putting integer “ j ” right after the integer “ i ” in the permutation. I call this “adjacency relation” pheromone model. This pheromone model is usually used for the class of sequencing or scheduling problems, like the travelling salesman problem (TSP).

CHAPTER 4. Algorithm Development Environment for Permutation-based Problems

Figure 4-10. Construction of Ants

Figure 4-11 and Figure 4-12 show the structures of block *Pheromone Initialization* and *Pheromone Update* respectively. I omit the detailed introduction because they are very fundamental ACO procedures and can be found in [47, 50, 56, 63, 72-78].

Figure 4-11. Pheromone Initialization

Figure 4-12. Pheromone Update

Chapter 5 Algorithms Configuration and Generation

With the module-based procedural library, which is managed by the LVRP tree structures introduced earlier, users are free to explore various aspects of algorithm design. From Figure 4-3 to Figure 4-8, it is shown that, the possible “features” associated to an algorithm that is being configured are located as the leaf nodes in the corresponding sub-trees. This means that the task of determining a certain “feature” of an algorithm by a user is essentially the process of finding a path from the root node to a particular leaf node in the corresponding tree. Therefore the task of configuring a complete algorithm becomes a process of traversing the trees to find a set of paths starting from the root nodes to the leaf nodes. The configuration of the algorithm is carried out along with the traversal of the tree(s). Therefore it is very natural to think of using the depth-first traversing in this work. Although breadth first traversing can be used (it can also find all the root-leaf paths), the process of the traversal is not as intuitive as that of depth-first traversal.

To achieve the depth first traversal specific for this LVRP tree, a LVRP traversal rule is defined as shown in Figure 5-1. The basic idea is to traverse only one child

of a variation node and every child of a property node. This rule is recursively applied to every “real” node in the tree.

```

LVRP_Traversal(NODE)
{
    Traverse to the variation (left) child of NODE: NODE.variation;
    choose only one child of NODE.variation: NODE.variation.OneChild
        LVRP_Traversal(NODE.variation.OneChild);
    Traverse to the property (right) child of NODE: NODE.property;
    for every child of NODE.property: NODE.property.childi
        LVRP_Traversal(NODE.property.childi);
    end for
}

```

Figure 5-1. LVRP Traversal Rule

Figure 5-2. Configuring Offspring Procedure by Traversing the Tree

Figure 5-2 illustrates the flow of a sample configuration of the component *Offspring Producer*. Starting from the root, it first traverses the variation (left) subtree of *Offspring Producer*. According to the left variation-right property traversal rule, only one variation, for instance Crossover, is chosen. It then continues to traverse the corresponding subtree (Crossover). Due to the depth-first rule, it traverses the variation (left) subtree of Crossover first. Assuming that the user chooses the Uniform crossover, it reaches the first leaf node, and continues to traverse the property (right) subtree of Crossover. Associated to it is a real number parameter Crossover Rate. After specifying the real value parameter for Crossover Rate, the flow proceeds to the second leaf node and goes on to traverse the property (right) subtree of *Offspring Producer*. From here on, only the Parent(s) Selection feature remains to be configured. To do this, it starts by traversing the variation (left) subtree of the Parent(s) Selection. Assuming that the user chooses Tournament out of the three variations of Parent(s) Selection available, it reaches the third leaf node. Since the property (right) subtree of Parent(s) Selection is empty, the traversal for this tree and the configuration of this component is accomplished. The traversal process covers three paths from the root node to three leaf nodes, Uniform, Crossover Rate and Tournament. This way, the user configures a feasible *Offspring Producer* instance, which is a form of uniform crossover with tournament parent selection methodology and a real number

crossover rate. The configuration of other components of the algorithm can be similarly carried out to achieve a complete genetic algorithm.

It is worth noting that with this novel hierarchical tree structure, system developers are involved only in high level user-system interactions via a graphical user interface in the process of configuring algorithms to address a specific problem area. The variations of any genetic algorithm can be represented by a flowchart, which is derived based on the blocks from the six components described earlier in Section 4.2. The configuration of each component can be carried out with the tree structure visualization. With such an approach, users with modest algorithm design expertise are able to configure and design various genetic algorithms through very high level and intuitive graphical interface. This way, the efficiency of algorithm design is greatly enhanced.

5.1 Code Generation

After configuring the desired genetic algorithm, a user can use the codes generator of ADEP to generate the C++ source code files based on the configuration he/she specified. ADEP adopts an object-oriented framework to facilitate code generation. The basic implementation of the code generator is illustrated as follows.

CHAPTER 5. Algorithms Configuration and Generation

Each “real” node in the tree structure is either procedure or parameter. The “real” node is regarded as an object (even for the real number parameter since it may have properties like range, precision, etc.) and thus is implemented as a class.

For each procedure node, the functionality of the procedure is implemented as a virtual function of the class. The exact function body of it can be determined by the choice of its “variation”. Take the *Crossover* as an example. The functionality of Crossover is always to take several parent individuals as the input and usually output one child (it can be more than one child). Once a variation version of *Crossover* is chosen, e.g., order-1 crossover, the function body is loaded as that of the order-1 crossover, which also takes several parent individuals as the input and outputs one child.

On the other hand, this virtual function may invoke some sub-routines or have some parameters, which are represented as “properties” of the node. These properties are implemented as the member variables of the node so that it can use these properties at any time. Take *Crossover* as an example again. *Crossover* function always needs to invoke a function called “parents selection”, which is implanted as the virtual function of class *Parents Selection*. Implementing an instance of *Parents Selection* within the *Crossover* class enables it to invoke the “parents selection” function at any time.

This way, the user-configuration information is stored in the tree structure. Based on the user-define configuration, i.e., the set of paths from root to leaves, the chosen classes along the paths are copied into a new program in term of .h and .cpp files. From the description in the last paragraph, it is guaranteed that the group of classes can form a feasible program.

It is worth noting that compared with borrowing procedures from some well-known libraries. The code generator described here saves a lot of effort for interfacing the library files to the user's own project. After that, the remaining effort is to customize the program to adjust the algorithm to solve various problems. I will describe in the next subsection how to customize the meta-heuristics program codes for solving different domain specific problems.

5.2 Customizing Genetic Algorithms

Figure 5-3. Components of GA

As shown in Figure 5-3, any genetic algorithm can be divided into three components from a structural viewpoint; genetic representation, genetic operations and fitness function. My work focuses on the permutation-based optimization problems. The artificial chromosomes coded as permutation strings are generated automatically. All the genetic operations, like crossover and mutation, are specifically designed for handling permutation strings. These operations tend to be standard features common to most GA-based problem-solving. They are usually problem-independent, though different operations may lead to performance that varies with problems. Therefore generation of the generic operations, which is usually the most time consuming of the coding effort, is carried out automatically based on the user's high-level specifications. Only the fitness function involves the problem's domain-specific information, and thus is problem-dependent. Fitness function is used to evaluate individuals based on the domain-specific knowledge of the problem being addressed. It plays a crucial role in guiding the GA to explore the entire solution landscape. It is also crucial in guiding the algorithm towards convergence to the optimum. The domain-specific information of a given problem can be exclusively stored in a special generated class, namely *problem*. In principle, one only needs to manually implement the fitness function in the generated genetic algorithm module to customize it for solving a particular domain of problems. It is noted that, the blank module of the fitness function is also generated as a source codes placeholder. The user only needs to overwrite its

function body to customize the algorithm for different problems. This way, the most time-consuming codes generation job in algorithm development is avoided. The effort for codes modification and customization is also minimized. Together with the high-level configuration process of algorithm design described earlier, it enhances the efficiency of the algorithm development significantly.

5.3 Customizing ACO

The customization of ACO is a little complicated than that of GA. Besides the definition/formulation of objective function, users also need to define the pheromone trace τ_{ij} following either an “allocation” model or the “adjacent relation” model.

Take facility location problem as an example for “allocation” model. In ACO for a problem instance with n facilities and n positions, it maintains an $n \times n$ “allocation” pheromone matrix $PH[\tau_{ij}]$. τ_{ij} indicates the desirability of facility i being located at position j . Based on these pheromone values, an artificial agent called “ant” is able to build a solution in a probabilistic manner. By evolving the pheromone matrix, ACO converges to some dominating τ_{ij} values. Statistically speaking, these dominating τ_{ij} form the optimal or very good resolution. In Chapter 7 and 8, an example of facility location problem, namely Quadratic Assignment Problem (QAP) will be presented.

Take Travelling Salesman Problem (TSP) as another example for “adjacent relation” model. In ACO for a problem instance with n cities, it also maintains an $n \times n$ “adjacent relation” pheromone matrix $PH[\tau_{ij}]$. τ_{ij} indicate the desirability of traveling from city i to city j . Based on these pheromone values, the artificial ant is able to construct a circuit (a solution). By evolving the pheromone matrix, ACO converges to some dominating τ_{ij} values. Statistically speaking, the circuit constructed based on these dominating τ_{ij} is the optimal or very good resolution. In Chapter 6, a general version of TSP, namely Vehicle Routing Problem (VRP) will be presented.

The formulation of τ_{ij} has a significant impact on the algorithm performance and should be decided to suit the specific domain of the problem. Unlike GA, the meaning of the pheromone trace in ACO must be defined by users based on the specific problem-domain knowledge. The rest of the ACO procedures, like pheromone update and initialization are standard and thus can be generated automatically.

5.4 Customizing Local Search

The customization of local search procedures is very similar to that of genetic algorithm. In principle, users only need to define the objective function to determine the change in objective value as a result of movement within the

CHAPTER 5. Algorithms Configuration and Generation

neighbourhood. The other procedures like definition of the movement, tabu rules and so on, are standard procedures and thus can be generated automatically. However, it is worth noting that because the neighbourhood movement in local search usually only modifies the solution marginally, in many cases, an efficient calculation of the change in objective value is possible [79].

Chapter 6 Case Studies

ADEP aims to deal with various combinatorial optimization problems with permutation string solution representation. In this chapter, several case studies of using ADEP to configure and generate genetic algorithms or hybrid genetic algorithms for solving several combinatorial optimization problems are presented. As described in earlier chapters, the idea is to harness the power of the code generator to produce source codes for a genetic algorithm automatically. In principle, one only needs to modify the objective function/fitness function to convert the codes for solving the target problem, without touching any main parts of the program, e.g., crossover, mutation, parents selection, fitness scaling, population update, etc. This way, compared to the conventional algorithm development method, users concentrate on the high-level aspect of algorithm design and create the codes of the algorithm for solving a particular problem with significantly reduced effort. Therefore the efficiency of the algorithm development process and the problem-solving capacity of the user are enhanced greatly.

ADEP was developed using Visual C++ 6.0. The latest version has about 20K lines of source code from scratch. The following case studies were carried out on an NEC 2.4 GHz PC with 512 MB memory.

6.1 *N*-Queen Problem

6.1.1 Problem Description

The *N*-queen problem is a classical combinatorial optimization problem. In recent years, it has found many practical applications including VLSI routing and testing [80]. Due to its simple and regular structure, it also has been frequently used as a testbed to develop new search algorithms. The methods proposed to solve the *N*-queen problem include genetic algorithm [81], evolution search algorithm [82] and local search [83]. The idea of the *N*-queen problem is to locate *N* queens on an $N \times N$ chessboard in mutually non-attacking positions. Generally, for $N > 3$, there are more than one solution. A possible solution of *N*-queen can be represented as a permutation string of length *N*, in which, any integer (from 1 to *N*) represents the column position of a queen on the chessboard, while the index of the integer represents the row position of the queen. There are $N!$ possible permutations for a problem with size of *N*. *N*-queen problem belongs to the constraint satisfaction problem (CSP) and it is NP-hard [84]. Table 6-1 shows the solution density with respect to problem size *N* [82].

Table 6-1. N -Queen Problem Solution Density

N (problem size)	Number of solutions	Search space ($N!$)	Solution density (per 10^6)
4	2	24	83333
5	10	120	83333
6	4	720	5556
7	40	5040	7937
8	92	40320	2282
9	352	362880	970
10	724	3628800	200
11	2680	39916800	67
12	14032	479001600	29

From the table, it is clear that, as the problem size N increases, the solution density reduces dramatically, and thus, the level of difficulty in finding a solution increases significantly.

6.1.2 Objective and fitness function

In my work for the N -queen problem, I simply count the number of the pairs of queens that attack each other. The goal is to minimize the number of collision pairs. Based on this formulation of the objective function, the N -queen problem is a constraint satisfaction problem. Therefore the fitness value used in the program for the N -queen problem can be simply set as the reciprocal of the objective value of the corresponding chromosome. It is not difficult to understand that the formulation of the objective and fitness function mentioned above only involve several lines of source code. In principle, this is all the coding that the user needs to deal with.

6.1.3 Algorithm Configuration, Generation and Customization

In my work, I configured and generated a hybrid-GA program described as shown in Figure 6-1 for solving the N -queen problem and the two problems discussed later in this section.

First, an initial population of chromosomes representing possible solutions is randomly generated. After evaluation, a population of offspring is generated by crossover and mutation operations with pre-specified probabilities. Then for each child, a local search optimization procedure is applied. This is a form of local search to augment the GA. Finally, the whole population is replaced by the offspring except for a pool of elites which survive into the next generation.

Figure 6-1. Hybrid Genetic Algorithm (adapted from [C3])

To further improve the performance of standard GA, I adopted the *2-opt* local search embedded in the algorithm. It repeatedly swaps the two genes in the

chromosome and accepts this movement if it leads to any improvement in the objective evaluation. In my work for the N -queen problem, I simply calculate the change in number of collisions resulting from a two-gene swap. Considering the fact that the 2-gene exchange operation causes only very localized modification of the chromosome, a simple speed up technique for calculating the objective value difference before and after the swap was applied. It counts only the changes of collisions that involve the two genes being swapped and thus requires minimal modifications of the code (several lines).

From the description above, it is clear that, with the aid of ADEP, developing an algorithm to solve N -queen problem is much simplified. Most of the codes can be generated automatically. In contrast, the conventional way of coding the entire algorithm from scratch is tedious and inefficient.

6.1.4 Simulations

I simulated the resulting algorithm on problems with N ranging from 100 to 2000. The code was compiled with Visual C++ 6.0 and run on an NEC 2.4GHz PC with 512 MB memory. The simulation results are presented in Table 6-2.

Table 6-2. Simulation Results on N -Queen Problem

N (problem size)	Time for finding the first solution (seconds)	Time (CPU time) for finding alternative solution (seconds)
100	0.08	0.15
200	0.50	0.87
300	1.80	3.19
400	2.25	3.38
500	4.97	6.60
600	6.84	13.06
700	9.21	17.73
800	12.95	19.97
900	14.95	16.22
1000	29.61	59.09
1500	70.09	132.42
2000	137.04	238.68

From the table, it is clear that, the resulting algorithm is able to uncover a solution for the N -queen problem with reasonable computational cost. For a very large problem with size of 1000, it took less than 30 seconds to generate the first solution. It is worth noting that although other methods like local search in [85] might be even faster in finding the first solution, the hybrid-GA can search and produce multiple solutions in parallel.

Table 6-2 shows that the hybrid-GA is able to produce the alternative solution of the problem in less computational time than what is needed for finding the first solution. For some applications, the capability to generate multiple and diverse solutions can be a useful feature from a practical point of view. Furthermore, the main objective of the easy configuration and code generation in ADEP is the quick and easy prototyping of a practically decent algorithm for users with only modest algorithm design background. In Chapter 7 and 8, an automatic configuration

engine in ADEP will be presented. It addresses the need for state-of-the-art algorithms. However, with the aid of this quick prototyping of ADEP, the trial-error cycle for tuning an algorithm can be significantly reduced, which may help the user to design a state-of-the-art algorithm.

6.2 Site Visitation Sequencing Problem for a Single Unmanned Reconnaissance Aerial Vehicle (URAV)

6.2.1 Problem Description

Unlike an unmanned combat aerial vehicle of which the ability to cover several targets in a single sortie is limited by the payload it can carry, unmanned reconnaissance aerial vehicle (URAV) can cover a large set of spatially distributed sites that are present within its site envelope. Given S being the set of reconnaissance sites that the URAV should survey, it is a general requirement that a URAV visits the useful vicinity of (if possible) each site exactly once in S , and lands back safely to the take-off zone or one of the other benign landing zones that are within its flight envelope. If the URAV returns to the take-off site, the problem becomes one of finding the *Hamiltonian Cycle*. Otherwise, I call the solution the longest path LP between take-off and landing sites. Obviously, the solution of this problem can be represented by an integer permutation, which represents the visitation sequence of the URAV. Therefore it is possible to be solved by ADEP.

An acceptable solution to this problem is essential in choosing “minimal risk” waypoints for the URAV flight path [86-88].

6.2.2 Objective and Fitness Function

In my work for the site visitation sequencing problem, I simply calculate the length of the longest substring/path in the chromosome by checking the connectivity between adjacent genes in linear time. The goal is to maximize the length of LP . The fitness value used in the algorithm for the N -queen is the reciprocal of objective value of the corresponding individual. However the program for site visitation sequencing problem is a maximization problem based on the objective evaluation. Therefore I simply set the fitness value the same as the objective value. From the descriptions above, it is not difficult to understand that the formulation and implementation of the objective function involves only several lines of code. In principle, this is all the coding that the user has to deal with.

6.2.3 Algorithm Configuration, Generation and Customization

I configured and generated a hybrid-GA program similar to the one in Figure 6-1 to solve this problem. Similar to that for the N -queen problem, I applied the 2-exchange local search in the algorithm for site visitation sequencing problem. It iteratively applies the 2-gene swapping and accepts this movement if it lengthens the longest successive segment in the chromosome.

6.2.4 Simulations

I tested the resulting algorithm on a dataset, which comprises 14 sparse graphs with edge densities (E_d) ranging from 0.0398 to 0.1947. These graphs simulate a set of reconnaissance sites and an airbase interconnected by a set of possible ‘safe’ paths. All 14 graphs in their graphic and adjacency matrix representations are available online [89]. An NEC 2.4GHz PC was used. Out of the 14 graphs, 7 are Hamiltonian while the remaining 7 are non-Hamiltonian. There are 10 graphs derived through special modifications to the Tutte graph [90]. Table 6-3 shows the experimental results of simulation on these graphs. Each graph is labeled as $G(a,b)$ where a is the number of nodes and b is the number of edges.

Table 6-3. Simulation results for site visitation sequencing problem

Graphs	Avg. CPU time (s)	Avg. longest path	Longest path obtained	Max. known path length
G(15,20)	0.08	14.0	14	14
G(20,30)*	0.11	20.0	20	20
G(20,37)*	0.10	20.0	20	20
G(46,69)	1.43	45.0	45	45
G(46,71)*	1.15	45.3	46	46
G(46,93)*	1.53	45.7	46	46
G(49,96)	0.98	48.0	48	48
G(49,142)	0.96	48.0	48	48
G(100,197)	4.29	95.5	98	99
G(100,197)*	5.29	97.8	100	100
G(100,203)*	4.17	96.6	99	100
G(100,207)	6.88	97.4	98	99
G(100,217)	9.75	97.9	99	99
G(100,217)*	6.23	98.2	99	100

*Hamiltonian graphs

In the table, it is shown that, for problems with size not larger than 49 nodes, which is already “very large” for many real applications, it is basically not a problem for my algorithm to uncover the longest known path or Hamiltonian cycle within 1.6 seconds of computational time. For larger problems with 100 nodes, the resulting hybrid-GA showed respectable performance in uncovering solutions that are very close to the longest path or Hamiltonian cycle, and the computational time is less than 10 seconds. For all the 14 graphs tested, the solutions with maximal length or $\{\text{maximal_length}-1\}$ were found.

6.3 Vehicle Routing Problem (VRP)

6.3.1 Problem description

The applications of vehicle routing problem (VRP), which was first proposed by Dantzig and Ramser [91] in 1959, are very common in real life. It can be described by the scenario that follows. Consider a depot having a fleet of vehicles with limited capacities and a set of customers, each with a certain demand for the merchandise or goods to be dispatched. The problem is to determine optimal routes for each vehicle to visit every customer exactly once in order to fulfill the demand. The most common goal for optimization is to minimize the overall distance traveled by the vehicles. An illustration is shown in Figure 6-2 where there are 3 vehicles and 8 customers or stations in the delivery network. The goal of optimization is to route all 3 vehicles such that all the 8 stations are visited by

one of the vehicles. The objective for routing the 3 vehicles is such that the total distance traveled by the vehicles is minimal. Figure 6-2 presents an illustration of one possible plan of routing the three vehicles starting and ending at the depot. It should be emphasized that in reality, for many real-life problems, it is common for practical constraints or limitations to be considered in the process of optimization.

Figure 6-2. Illustration of a 3-Vehicle Routing Problem

A common variation of the VRP can be derived by considering the limitation in the capacity of each vehicle. This is referred to as the capacitated VRP (CVRP). Another variation of the VRP considers the scenario where some or all the customers are to be supplied within a certain time window. This is referred to as the VRP with time windows (VRPTW). In minimizing the overall distance traveled, the derived solution also must not violate the specified constraints. It is clear that after clustering the customers according to the vehicle that services them, the routing of each vehicle is essentially the famous traveling salesman problem

(TSP) [2]. This is why the VRP is sometimes regarded as a generalization of TSP, and is therefore NP-hard. It is considered as one of the most difficult combinatorial optimization problems known.

Due to the practical relevance of VRP and its considerable difficulty, it is therefore not surprising that the VRP continues to be a subject of intense study in the past few decades. Many solution techniques have been proposed to solve the VRP. From literature survey, one can classify the methods into exact, heuristics and meta-heuristics approaches. For example, in [92], Fisher proposed a branch and bound algorithm, which is an exact enumeration approach for solving VRP. Although it succeeded in solving a problem with 71 customers to optimality, it failed to solve some smaller instances. Due to the NP-hardness of VRP, this approach can only handle the instance with up to 100 customers. This is why heuristics and meta-heuristics were proposed for solving VRP with approximate solutions in reasonable time. The examples include 2-phase algorithm [93], tabu search [94], ant algorithm [95, 96] and genetic algorithm [97-100].

My work considers the formulation for a CVRP, a variant of the VRP whereby the capacity of each vehicle is limited, although it should be emphasized that the CVRP can be subjected to further practical constraints in real-life supply chain management. I use the ADEP to create a hybrid genetic algorithm as the problem-solving algorithm. I tested my algorithm on a class of well-known CVRP

benchmarks and a real-life CVRP application, namely the fuel truck dispatch system (FTDS). The scenario of the FTDS describes the routing of a fleet of dispatch trucks for distributing fuel to refueling stations in Singapore.

6.3.2 Solution Representation

In my work, I use ADEP to configure and generate a hybrid genetic algorithm for solving the VRP. It is generally known that the scheme for genetic representation of the solution i.e., the chromosome coding structure, can play a crucial role in GA. Consequently, this may have a profound impact on the algorithm's performance significantly. ADEP enables use to study different genetic representations and their impact on the performance of the algorithm very conveniently. I study two forms of genetic representations used in the hybrid genetic algorithm for solving the VRP and analyze their influence on the performance of the algorithm.

Consider a fleet of trucks $\{T_1, T_2, \dots, T_m\}$ and a set of stations $\{S_1, S_2, \dots, S_n\}$ to be visited, where m is the number of trucks and n is the number of stations (excluding the depot station) to be visited by these trucks. A possible routing plan in terms of chromosome can be encoded by either of the two representations introduced as follows.

6.3.2.1 Genetic representation 1

In the first solution representation, I encode the CVRP solution as an integer

permutation string (the chromosome in GA) $C = \langle \lambda_1, \lambda_2, \dots, \lambda_{(m+n)} \rangle$, where $\lambda_i \in \{1, 2, \dots, (m+n)\}$. Thus I can use the ADEP to generate the modular codes for the solution methodology. The genes of each chromosome decode into two types of information, index of the station identity or index of the identity of a truck. Eq. 6-1, which is used to determine α , is applied during the decoding.

$$\alpha = \begin{cases} \lambda_i & \text{if } \lambda_i \leq n \\ \lambda_i - n & \text{if } \lambda_i > n \end{cases} \quad \text{Eq. 6-1}$$

If $\lambda_i \leq n$, the value α indexes a station while $\lambda_i > n$ results in α indexing the identity of a truck. With this representation, the information on both the stations and the trucks are embedded in a single chromosomal structure. The decoding of a given chromosome is carried out by building up a sequence of visitation sites for each truck starting circularly from the left to right. A truck denoted by the corresponding *truck indicator* is assigned to visit the stations that follow until the next *truck indicator* delimits the sequence. Figure 6-3 shows the first solution representation scheme (I named it genetic representation 1, GR1 for short) for $m=3$ and $n=7$ for illustration.

Figure 6-3. Genetic Representation 1

The genes with alleles from 1 to n represent the stations that the trucks need to cover. The remaining genes with alleles from 8 to 10 identify the trucks. For example, “8” represents the first truck, which is T_1 . Similarly, “9” represents T_2 and “10” represents T_3 . In Figure 6-3, T_1 is followed by station “3” and “4” before the next truck T_2 . This means that T_1 visits stations “3” followed by “4”. This chromosomal structure not only indicates the clustering information but also specifies the visitation sequence for each truck within the decoding process. This allows good clustering information as well as good segment of visitation sequence to be inherited effectively even through standard genetic operations for permutation-based GA, like Order-1 crossover, which favors the transfer of sequencing information during evolution.

6.3.2.2 Genetic representation 2

I extend GR1 to derive a second genetic representation scheme. Similar to GR1, the GR2 scheme encodes the solution of VRP as an integer permutation string $C = \langle \lambda_1, \lambda_2, \dots, \lambda_n \rangle$, where $\lambda_i \in \{1, 2, \dots, n\}$. However the length of the chromosome is n instead of $(m+n)$. The alleles of genes are the stations identity. In this representation, the demand of the customers is also taken into account in the decoding process. Given a list of the demand of the stations $\{D_1, D_2, \dots, D_n\}$. The decoding of a given chromosome is carried out from the left by building the visitation sequence for each truck, meanwhile satisfying the capacity constraints. Each selected truck is assigned to service the left most station, which is not assigned yet, and its subsequent stations one by one, until one more station added into the visitation sequence will lead to the total demand of these stations to exceed the capacity of the truck. Then the next selected truck is assigned through the same process until all stations are covered. The selection order of the trucks is carried out as follows. Given the list of the capacity of trucks $\{C_1, C_2, \dots, C_m\}$ and delivery amount of trucks $\{A_1, A_2, \dots, A_m\}$ (note that A_i varies during decoding). I selected the truck T_i , which maximizes the ratio A_i/C_i , to be the next candidate truck for assignment. In other words, the truck that utilizes its capacity maximally will be selected with priority. Figure 4-7 illustrates the same scenario presented in Figure 6-3 while using GR2. The ratio A_i/C_i for each truck is shown in the parentheses following the corresponding *truck indicator*. In Figure 6-4, since truck T_1 utilizes 100% of its capacity to cover the first two stations, it is firstly selected for

assignment. After that, T_2 utilizes 35/47 of its capacity to cover station 3 and station 4, while T_3 can utilize 35/45 of its capacity to service the same stations. Therefore T_3 is selected to service these stations and the rest of the stations are covered by T_2 .

Figure 6-4. Genetic Representation 2

As an extension of GR1, GR2 simplifies the chromosomal structure by hiding the truck information. In GR1, since the chromosome strings are initialized randomly, this probably leads to infeasible solutions which violate the constraints. In contrast, GR2 not only specifies the clustering and visitation sequence of stations, but also optimizes the truck assignment with consideration of capacity constraints during decoding. This contributes to a high chance for GR2 to produce feasible solutions for VRP. Similar to GR1, good clustering features as well as good segment of

visitation sequence of chromosomes in GR2 can be effectively inherited through standard genetic operators.

6.3.3 Objective Function and Fitness Function

The objective of the search is to minimize the overall distance traveled by all the trucks. Given that a truck has V number of stations to service, and the visitation sequence is $\{S_1, S_2, \dots, S_V\}$, the distance traveled by a truck T_i is calculated as follows:

$$\text{dist}(T_i) = \delta_0 + \sum_{i=1}^{V-1} d_{i(i+1)} + d_{01} + d_{V0} \quad \text{Eq. 6-2}$$

In the above equation, $d_{i(i+1)}$ ($1 \leq i \leq V-1$) indicates the distance between station S_i and S_{i+1} in the tour sequence of the truck. d_{01} is the distance from the depot to station S_1 while d_{V0} is the distance from station S_V to the depot. δ_0 is a constant value denoting the distance covered by the truck at the depot, such as departure and parking, which in my case is set to zero. Let R denote the routing plan for a fleet of vehicles decoded from a genetic string. The objective function for the search can be written as Eq. 6-3 below:

$$\text{dist}(R) = \sum_{j=1}^T \text{dist}(T_j) \quad \text{Eq. 6-3}$$

In the above equation, $\text{dist}(T_j)$ returns the distance traveled by the j -th truck, and $\text{dist}(R)$ represents the total distance traveled according to plan R . From the distance

computed, the raw fitness value of a chromosome that results in a feasible routing plan can be calculated as follows:

$$raw_fitness(R) = \frac{W}{dist(R)} \quad \text{Eq. 6-4}$$

where W represents a numerical constant.

It is noted that with such formulation of the fitness evaluation function, the algorithm does not exclude infeasible solutions during the search. If the routing plan is such that there is insufficient fuel for any trucks to fulfill the demand of all the stations it is to service, the resulting plan is deemed to be infeasible. To handle such situations, I incorporate penalty into the fitness function. The penalty applied to an infeasible solution is proportional to the extent of the shortfall of fuel for the corresponding truck. Eq. 6-5 represents the overall fitness function for the genetic algorithm incorporating the penalty component for infeasible solutions.

$$fitness(R) = raw_fitness(R) - \mu \sum_{j=1}^T \frac{shortfall(T_j)}{capacity(T_j)} \quad \text{Eq. 6-5}$$

In Eq. 6-5, $shortfall(T_j)$ represents the extent of shortage for the j -th truck with respect to the overall demand of the stations that it has to service, and $capacity(T_j)$ represents the capacity of the j -th truck. μ is the weighting factor for the penalty to be applied.

I experimented with algorithm that configures only feasible solutions during the search. This is akin to complete penalty for infeasible solutions. Compared to

applying complete penalty, i.e., discarding chromosomes that result in plans that are not feasible, the simulations have shown that using the fitness function as shown in Eq. 6-5 offers two advantages. First, it preserves those chromosomes with good objective values although infeasible, especially during the early stages of the search. This way, their good “attributes” are therefore inherited by their offspring, increasing the chances of producing good and feasible solutions. The second advantage of using penalty incorporated fitness function is one of direct practical relevance. For situation where there is no feasible solution or the algorithm fails to produce any feasible solutions, solutions with the lowest distance although infeasible can still be generated. These solutions are still meaningful and useful, especially for further evaluation in light of the practical requirements of the customers.

It is worth noting that GR1 operates in a totally blind manner. Therefore the stochastic genetic operations like crossover and mutation are very likely to produce infeasible solutions if they are encoded and decoded using GR1. The impact of the penalty value (the second term in Eq. 6-5) to the entire fitness value is very sensitive to the setting of the weighting factor μ . As a result, a delicate fine-tuning of the value of μ is required to tune the entire algorithm to its best performance.

In contrast, GR2 takes the capacity constraints into account, which coordinates and optimizes the truck distribution to some extent during the encoding and decoding.

Therefore the chance of producing infeasible solutions of GR2 is much less than that of GR1. In other words, the value of the penalty is more likely to be zero or very small even if μ is very large. As a result, the impact of the penalty to the entire fitness value and consequently the performance of the entire algorithm is much less sensitive to the setting of the weighting factor μ in GR2 compared with that in GR1. This can be perceived as another advantage of GR2 over GR1.

6.3.4 Local Search for TSP

In Eq. 6-2, the calculation of the shortest distance traveled by a truck is essentially the same as a traveling salesman problem (TSP) [2]. TSP is a very famous class of combinatorial optimization problems and is itself NP-hard. The general TSP involves finding the shortest circuit for a given set of nodes to be visited. In my approach, the accuracy and efficiency of the scheme for solving this routing problem has a great influence on the effectiveness of the global algorithm. It involves enumerating all the $(n-1)!/2$ possible routes to determine the sequence of visits that result in minimum distance, where n is the number of stations for a truck to visit. Although the exhaustive method guarantees the optimal solution, the computational complexity is in the order of $O(n!)$. As n increases, the computing time tends to increase exponentially.

As an alternative, local search methodology is able to produce the near-optimal routing sequences for each truck in manageable time. Successful examples of local

search techniques for TSP optimization can be found in [43]. In my implementation in the ADEP library, I simply apply the 2-opt local search (the version for TSP) with “first improvement” policy. It iteratively chooses a segment of the visitation sequence and reverses the order of it. Any improvement as a result of this “movement” is accepted. The process is repeated until no improvement is possible by such “movement”.

6.3.5 Algorithm Configuration, Generation and Customization

I used ADEP to configure a hybrid genetic algorithm (hybrid-GA) similar to the one in Figure 6-1 except for the local search version. I generated the source codes and customized the automatically generated program to solve the VRP by modifying the objective function and fitness function as described earlier in Subsection 6.3.3., without touching the other parts of the codes.

6.3.6 Simulations

I tested my hybrid-GA using the two genetic representations on the CVRP instances for a real-world application, namely Fuel Truck Dispatch System (FTDS). I also tested them on some well-known benchmarks of CVRP and compared their performance. The simulation results are presented and analyzed.

6.3.6.1 Comparison on CVRP benchmark instances

I first compared the hybrid-GAs with the two different representations on a well-known CVRP benchmark datasets [101]. I tested the algorithms on two representative instances with 50 and 75 stations. 10 independent trials were carried out to evaluate the average performance. All simulations were done on a NEC PC with 2.4GHz CPU clock speed. The results of simulations using different representations are presented in Table 6-4 and Table 6-5. The parameters of the algorithms for my simulations are as follow:

Population size = 30;

Crossover rate = 0.9;

Mutation rate = 0.1;

Number of generations = 300;

Time bound = 30 seconds.

Table 6-4. Comparison on a 50-station benchmark problem

Trial	Distance (Km)	
	GR1	GR2
1	525.93	524.61
2	527.98	525.93
3	534.82	526.93
4	524.81	524.81
5	524.61	525.93
6	525.93	524.81
7	527.98	524.61
8	525.93	525.93
9	532.00	524.61
10	527.98	524.81
average	527.80	525.30

Table 6-5. Comparison on a 75-station benchmark problem

Trial	Distance (Km)	
	GR1	GR2
1	866.69	858.24
2	868.07	844.77
3	863.33	838.60
4	861.26	849.77
5	867.34	838.89
6	858.40	845.67
7	860.56	863.33
8	863.33	844.77
9	875.37	855.34
10	855.34	861.26
average	863.97	850.06

I applied the Wilcoxon rank-sum test with Bonferroni correction on the results presented in Table 1 and Table 2 for further statistical analysis. At a confidence level of 95% ($\alpha=0.05$, $n=10$), the probabilities of GR2 being better than GR1 are 99.41% and 99.85%. In other words, GR2 is significantly better than GR1.

6.3.6.2 Comparison on FTDS instances

To further validate the hypothesis that GR2 is better than GR1, I tested and compared the two algorithms based on a real-life CVRP application – Fuel Truck Dispatch System (FTDS). FTDS is designed to deal with the distribution of fuel for the purpose of replenishing the supply at refueling stations in Singapore. The system generates a plan for routing the various fuel trucks being managed. Each

truck dispenses a fixed amount of fuel allocated to the station. The main goal of planning is to minimize the total distance traveled by all the trucks. The two hybrid-GAs are simulated and tested on a FTDS dataset involving 30 refueling stations island wide, including the terminal station. Table 6-6 shows the 30×30 distance matrix. Each entry in the matrix corresponds to distance between any two stations. Since the matrix is symmetric, only one half of the matrix is shown. Excluding the terminal station, which in this case is station 2, there are 29 refueling stations to be serviced by a fleet of trucks. Typically at any time, not necessarily all the stations are involved in the planning as some may have ample supply and hence there is no need for replenishing of supply. Depending on the demand, it may not be necessary to deploy all the trucks at any one time. The hybrid-GA generates plans that optimize on the number (or type) of trucks for deployment based on cost consideration.

I simulated the scenario where the terminal station has up to 5 trucks for deployment. At any time, a station may submit a request for a certain amount of fuel. Under normal circumstances, the capacity of all the 5 trucks is known beforehand. It is worth noting that unlike standard CVRP problems as shown in [101], FTDS is required to handle scenarios with trucks having different capacities. Furthermore, the distance between stations is the real distance on road instead of the Euclid distance.

Table 6-6. Distance matrix in FTDS

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
1	x	5.2	16.3	19.0	24.3	28.7	27.8	30.8	33.6	42.2	31.5	27.3	38.0	14.2	9.1	16.2	14.5	19.0	4.7	15.9	19.9	28.4	28.1	31.5	33.3	18.4	25.1	38.6	31.7	23.2
2		x	6.7	9.7	14.9	18.0	27.5	20.2	22.5	31.1	22.1	22.5	25.9	11.5	11.7	12.2	5.6	7.3	7.6	6.5	10.9	17.3	15.4	20.4	18.9	6.5	13.3	27.5	21.1	13.8
3			x	3.2	7.1	11.1	17.1	13.3	15.6	24.2	13.3	15.6	20.0	10.3	15.5	10.6	6.8	4.5	14.2	1.8	3.4	10.4	10.5	13.5	15.3	4.8	7.6	20.6	13.7	7.6
4				x	8.2	9.1	15.1	11.3	13.6	22.2	12.4	13.6	18.0	9.0	13.8	10.1	6.4	6.8	14.5	5.3	1.4	8.4	8.5	11.5	13.9	3.0	5.6	18.6	11.7	8.8
5					x	11.6	18.3	10.2	11.6	18.8	7.2	16.1	14.6	17.2	23.3	16.7	15.3	7.1	21.5	8.6	5.9	6.4	11.3	9.5	8.2	11.0	8.6	15.2	7.5	2.0
6						x	6.8	6.6	7.0	18.5	11.7	4.6	17.3	11.8	18.7	15.5	15.0	14.4	23.2	12.9	7.6	7.7	2.5	6.5	12.6	11.3	4.4	17.9	11.04	13.1
7							x	11.5	12.2	19.2	16.6	3.6	19.6	10.2	17.1	14.0	17.5	20.7	23.6	19.2	13.9	12.6	7.1	11.4	17.5	17.6	9.1	20.4	15.9	20.2
8								x	1.9	12.6	5.7	12.7	10.2	18.4	24.5	17.9	17.4	15.5	25.7	18.1	9.4	4.1	5.0	0.8	6.6	13.7	2.9	12.1	5.1	13.2
9									x	11.8	7.2	12.3	10.5	19.4	26.3	19.4	18.9	17.0	27.2	19.6	10.9	5.6	4.7	1.3	7.6	15.2	4.4	12.1	6.6	14.7
10										x	12.1	19.9	3.5	27.0	33.9	29.3	28.8	25.7	37.0	26.5	19.8	15.4	15.8	11.0	10.9	24.9	17.2	2.5	11.8	20.2
11											x	18.2	7.1	5.1	26.1	19.5	18.9	14.4	28.8	16.0	8.2	3.6	9.6	5.8	0.6	14.9	7.8	8.8	1.8	8.8
12												x	18.5	8.7	15.6	12.5	18.5	19.6	24.0	17.1	12.1	13.5	7.0	11.0	17.0	15.4	9.2	20.0	15.1	21.8
13													x	24.7	31.5	24.2	23.5	21.4	31.7	20.5	14.2	8.8	14.2	8.5	5.7	19.6	12.4	0.9	7.3	17.0
14														x	4.9	1.1	5.0	14.3	10.9	10.7	12.4	16.2	14.2	18.0	19.7	6.5	13.3	27.3	20.4	20.4
15															x	6.6	8.9	18.8	8.0	15.8	18.9	22.7	23.1	25.8	27.6	12.1	19.5	32.9	26.0	23.0
16																x	5.3	14.6	12.9	10.9	11.9	15.7	15.0	18.1	19.2	6.7	12.7	25.2	18.4	18.2
17																	x	10.2	8.4	4.8	9.7	14.9	14.5	18.0	19.7	3.9	12.4	25.1	18.2	14.4
18																		x	16.3	3.5	8.0	12.1	13.9	15.9	15.0	10.1	11.8	22.9	13.9	6.1
19																			x	13.3	17.8	23.1	22.7	26.2	27.0	12.0	20.6	33.3	26.4	20.5
20																				x	7.9	14.7	12.3	14.9	15.1	4.6	10.2	22.0	15.1	7.7
21																					x	5.1	7.0	8.6	9.7	0.0	4.9	15.8	8.9	6.8
22																						x	5.6	3.5	4.3	11.0	3.8	10.4	3.5	7.8
23																							x	4.4	10.5	10.8	1.2	15.8	9.0	13.2
24																								x	5.5	14.0	4.3	11.2	4.7	12.8
25																									x	15.8	8.7	7.3	1.5	8.7
26																										x	7.4	21.1	14.3	11.1
27																											x	14.0	7.1	11.4
28																												x	8.5	17.8
29																													x	8.6
30																														x

Table 6-7 presents a sample of the simulation results for what I termed as a tight configuration scenario. In this situation, the overall capacity of the trucks deployed is just about enough to meet the overall demand of the stations. To simulate this, the overall demand of fuel by all 29 stations is equivalent to 97.6% of the overall capacity. A sample of the routing plan involving 5 trucks is depicted in Figure 6-5. The figure shows 30 stations spread across the island of Singapore. The routing of the 5 trucks is generated for a tight configuration scenario using GR2.

Table 6-7. Comparison on FTDS Instance

Trial	Distance (Km)	
	GR1	GR2
1	213.56	212.52
2	213.45	212.52
3	214.35	214.21
4	212.62	212.52
5	212.69	213.65
6	212.52	212.52
7	213.04	212.52
8	213.58	213.54
9	214.37	212.52
10	214.21	212.52
average	213.44	212.90

Figure 6-5. A sample routing plan on Singapore Island

I applied the Wilcoxon rank-sum test with Bonferroni correction on the data in Table 6-7. At a confidence level of 95% ($\alpha=0.05$, $n=10$), the probability of GR2 being better than GR1 are 99.61%. It confirms that GR2 is significantly better than GR1.

6.3.6.3 Comparison with other algorithms

I tested the hybrid-GA with GR2, which showed to be the better of the two representations, on several datasets of CVRP benchmarks [101] and compared it with some other competing VRP methods, in particular, the simulated annealing

[102], the tabu search [103] and the hybrid genetic algorithm [100] referred to as OS, GHL and BB respectively. The results obtained were compared against the other approaches, as summarized in Table 6-8. In Table 6-8, entries in the first row identify the instances of the benchmarks and their sizes (i.e. the number of stations). In the next 3 rows, the best results in term of traveling distance obtained by these three competing methods are presented. In the fifth row, it reports the best and the average performance of my evolutionary algorithm and the last row shows the best-known values of the corresponding benchmarks.

Table 6-8. Simulations on benchmark problems

Benchmark name (n)			vrpnc1(50)	vrpnc2(75)	vrpnc3(100)	vrpnc4(150)	vrpnc5(199)
OS (best trial)			524.61	844.00	838.00	1044.35	1334.55
GHL (best trial)			524.61	835.45	829.45	1036.16	1322.65
BB (best trial)			524.61	835.26	827.39	1036.16	1324.06
GR2	Best	distance	524.61	838.60	830.70	1044.35	1334.55
		deviation	0.00%	0.40%	0.55%	1.55%	3.34%
	Average	distance	525.30	850.06	845.63	1064.52	1353.92
		deviation	0.13%	1.77%	2.36%	3.51%	4.84%
Best-known			524.61	835.26	826.14	1028.42	1291.45

According to the table, the results obtained by my algorithm are competitive. For the smaller benchmarks, the best solutions of hybrid-GA are comparable to the best solutions of the other approaches.

6.3.7 Performance Analysis

From the simulation results, the conclusion is drawn that GR2 outperforms GR1. GR2's superiority can be explained from a viewpoint of search complexity. GR2

incorporates the truck indicators within the chromosomal structure. Though this simplifies the decoding process, it complicates the problem unnecessarily by increasing the dimension of the problem from n to $(m+n)$. As a result, the size of the search space increases dramatically from $n!$ to $(m+n)!$. This may be the first reason why GR2 tends to perform better than GR1, even though they have very similar chromosomal structures. Another reason can be the fact that GR2 uses the problem information (the capacity constraints) while GR1 does not. Use of the problem domain information sometimes can narrow down the search space and avoid totally blind search.

GR2 offers another attractive advantage over GR1. In the under-capacity scenario (when the overall demand exceeds the overall capacity), besides the penalty function mentioned in Eq. 6-5, GR2 can be used to generate a dispatch plan, which handles the situation by assigning trucks for multiple trips between the depot and stations, not just simply providing an infeasible plan. During the decoding of GR2, once the overall demand exceeds the summation of the capacity of all trucks, one or even more “virtual trucks” are introduced after the “real” trucks to service the uncovered stations. Each virtual truck is associated to a real truck, which is scheduled to refuel itself after returning to the depot, and start out for the next trip to service the remaining stations. As shown in Figure 6-6, the total demand (155) exceeds the total capacity of the three real trucks (132). After finishing the assignment for all the real trucks, there are still 2 stations (S_8 and S_9) uncovered.

Therefore, a virtual truck which is associated to T_1 is introduced. Truck T_1 is thus scheduled to service S_1 and S_2 first. After refueling at the depot, it is sent out for its second trip to continue servicing S_8 and S_9 . This way, the algorithm can help operators to handle the under-capacity situation with more options that are useful practically. The operator does not need to consider each trip separately or to run the algorithm again to get a feasible dispatch plan. GR2 enables the algorithm to plan the entire dispatch schedule in single run.

Figure 6-6. Virtual Trucks Using GR2

6.4 Summary of the Case Studies

In this Chapter, several case studies of using ADEP to generate meta-heuristics have been presented. Based on these case studies, it is clear that ADEP addresses the need for easy and quick prototyping of a practical metaheuristic for users with only modest algorithm design background. With ADEP, the configuration of the algorithms is convenient and most coding task is done automatically. It saves a great amount of time for general users and especially enables those users with more than modest algorithm design background to concentrate on the high-level algorithm development.

The case studies on the N -queen problem and the visitation sequence problem showed that with only minimal efforts on the decoding function (just several lines of code for each problem) and without touching any other parts of the program, users can generate a well-performing hybrid-GA. Although the performance may not be as competitive as some state-of-the-art algorithms, they produced practically decent results. The need for well-performing algorithms will be addressed by the automatic configuration engine as presented later in Chapters 7 and 8.

The case study on VRP serves as an even better example. Without changing the encoding (permutation string), users were able to quickly explore different genetic representations. This speeds up the trial-and-error cycle and enable us to concentrate on the high-level algorithm exploration with different genetic

representations. Because of the increase in research efficiency, which enables us to explore more possible directions within a given time frame, a hybrid GA with competitive performance to some existing algorithms were finally developed.

Chapter 7 Automatic Configuration Engine

7.1 Motivation

The schemes described in earlier chapters facilitate the process of algorithm configuration and reduce the effort for tedious source codes generation and modification significantly for real-life applications of combinatorial optimization problems. This way the efficiency of the algorithm development is enhanced dramatically.

However, the methodology described in earlier chapters does nothing to improve the quality of the resulting algorithm for the specific class of problems being addressed. The performance of the resulting algorithm still depends very much on the experience and expertise of the user. Without profound knowledge on the meta-heuristics design or the specific knowledge about problem domain, one may still face difficulty in designing and configuring a well-performing algorithm.

To address the need for automatic algorithm optimizer, a novel evolutionary methodology was proposed and developed in ADEP as an automatic configuration engine to optimize the configurations of the meta-heuristic algorithm. Users or

system integrators with modest background knowledge and experience on meta-heuristic algorithms design may use the automatic algorithm configuration engine to automatically search for an optimal configuration of meta-heuristic algorithm (including optimized program structure, parameter tuning, procedure selection, etc.), for solving a class of combinatorial optimization problems.

7.2 Automatic Configuration Engine

To begin, users will first select some sample problems as the training set from the class of problems being addressed. The users may specify the expected performance of the algorithm beforehand, gauged by different performance metrics, e.g. efficiency, effectiveness, robustness, stability, etc. The automatic configuration engine has the capability to learn from the training set as well as the users' predefined requirements, adapting its internal context, iteratively training itself to produce algorithms with progressively better performance. Once users are satisfied with the performance of the best-found algorithm, they generate the source codes of the algorithm as a stand-alone program, which can serve as the problem-solving optimization algorithm for the class of problems used in training or other similar problems. This way, the effort for algorithm optimization and generation is minimized significantly. Not only the efficiency but also the quality of the algorithm developed and the problem-solving capacity of general users are enhanced dramatically. I tested the automatic configuration engine on a diverse set of benchmark instances of a well-known combinatorial optimization domain,

namely the quadratic assignment problem (QAP) [1]. It was shown that the automatic configuration engine was able to automatically produce algorithms with highly competitive level of performance.

As shown earlier in Figure 5-2, the process of configuring a meta-heuristic algorithm is essentially equivalent to the process of traversing the trees according to the LVRP traversal rules described earlier. The process of configuring the features of an algorithm component is equivalent to finding a path from the root to a certain leaf node. This way, the configuration process is mapped to a tree search/traversal problem.

It is worth noting that a logical way of uncovering the optimal configuration of an algorithm is to enumerate all the possible combinations of the root-leaf paths (for real number parameters, discretization was applied to avoid infinite search space and the range and interval can be user-defined). However, the time complexity of such an approach is not manageable. Therefore the automatic configuration engine was developed to address the need for quick and automatic configuration of the meta-heuristics for the class of problems being addressed. It is a desirable prerequisite that the algorithms configured through the process of evolutionary learning should be optimal or close to it.

In the process of finding a path from the root to the leaf node, the most important step is to make the decision on which branch to choose at every “traversal split node”. As described earlier in Chapter 5, the “variation” virtual nodes belong to the “traversal split nodes”. A decision in choosing a variation of the parent node is required at the traversal split nodes. Furthermore, users can assign a number of predefined candidate values to the real number parameters. Therefore the real number parameter nodes also belong to the “traversal split nodes”, since a decision on the specific value assigned to the parameter is required here. It is clear that since every sub-tree of a “property” node needs to be traversed, it does not belong to the “traversal split nodes”.

A trace value is assigned on each branch of a “traversal split node” to guide the decision-making. The traces can be viewed as the weights of each branch. Given there are n number of branches stemming from a “traversal split node” and $\tau_{(i)}^j$ ($i=1,2,\dots; j=1,2,\dots,n$) indicates the trace value of the j -th branch (choice) of the node at the iteration i . The decision of choosing a branch at iteration i is based on the trace values.

$$Choice(i) = DecisionMake(\tau_{(i)}^1, \tau_{(i)}^2, \dots, \tau_{(i)}^n) \quad \text{Eq. 7-1}$$

The function *DecisionMake* is normally a probabilistic function and it can have many variants. One possible form can be based on simple roulette wheel selection:

$$P_{(i)}^j = \frac{\tau_{(i)}^j}{\sum_{m=1}^n \tau_{(i)}^m} \quad \text{Eq. 7-2}$$

where $P_{(i)}^j$ indicates the probability of j -th branch being chosen at iteration i .

This way, the system is able to configure an algorithm by traversing the trees guided by the strength of the traces laid earlier.

```

Procedure Training for Configuration Optimization

  TraceInitialization(Trees);

  do

    for  $i=1$  to #PopSize

      AlgorithmPop[ $i$ ] = ConfigureAlgo(Trees);

      EvaluateAlgo(AlgorithmPop[ $i$ ], TrainingSet);

    end

    UpdateTrace(Trees, AlgorithmPop[best]);

    while (TerminationCondition() != TRUE)

      return (Trees)

    end

```

Figure 7-1. Automatic Configuration Engine - Training Phase

Figure 7-1 shows the flow of the training process of the automatic configuration engine. First, the traces in the trees are initialized. Each branch of a “traversal split node” (highlighted in bold) is assigned with an even trace value as shown in Figure 7-2. This means every possible configuration setting is awarded the same chance of being selected initially. Then in the evolutionary loop, it configures #PopSize, the number of independent algorithms by traversing the tree guided by the trace values. These algorithms are in turn evaluated by carrying out test runs on the training set, which consists of a number of sample instances of the target class of problems being addressed. After that, the trace values in the tree are updated based on the best-performing few algorithms in the current iteration. The above steps (except the trace initialization) are repeated until the termination condition is met. Finally the tree branches with the maximum trace values at each split node are output as the result of the training process.

Figure 7-2. Trace Initialization

Figure 7-3 illustrates the process of trace update in *Offspring Producer* as an example. Given that the system configures the best-performing algorithm $ALGO_{i+1}$ based on the trace information of iteration i and $ALGO_{i+1}$ contains the configurations of *Offspring Producer* as shown in terms of the three bold root-leaf paths in Figure 7-3. Then the corresponding trace values along these three paths are emphasized/increased. A more intuitive example of implementation of this update will be illustrated in Section 8.2.2. As a result, the relative values of the trace values along the three paths are increasing and the rest trace values, which are not covered by the three paths, are decreasing. This way, good configuration features are rewarded with a higher chance of being selected in the next iteration and this preference is accumulated in a probabilistic and statistical manner as the evolutionary process progresses. Finally, the evolutionary process converges with a few root-leaf paths dominating the trees by large relative trace values along them. These paths, i.e. the configurations, constitute a good algorithm derived from training based on the target class of problems being addressed. It needs to be verified by carrying out test runs on the set of test problems.

Figure 7-3. Trace Update

It is worth noting that users may have different requirements on the performance of the algorithm due to different real-life scenarios. These requirements and differences can be embodied in the way the system evaluates an algorithm (e.g., based on different measurement metrics). It is necessary to emphasize that the automatic configuration engine has the capacity to learn not only from the specific nature of the problem being addressed, but also the different user requirements on the algorithm. The domain-specific nature of the problem, together with the user requirements, guides the search in converging toward configurations suitable for the problem scenarios.

Chapter 8 Validation on Automatic Configuration Engine

To validate the usefulness and effectiveness of the automatic configuration engine, I tested it on a diverse set of benchmark instances of a well-known class of combinatorial optimization problems, namely the quadratic assignment problem (QAP). The algorithm derived from training was tested and compared to other two recently reported algorithms for the QAP.

8.1 Quadratic Assignment Problem

The quadratic assignment problem (QAP), which was first introduced by Koopmans and Beckmann [1] to solve a facility location problem, is a class of NP-hard problems. It can be described as follows. A problem of size n , can be represented by two $(n \times n)$ matrices $\mathbf{A} = [a_{ij}]$ and $\mathbf{B} = [b_{ij}]$. Matrix $\mathbf{A}$ can be interpreted as a distance matrix, i.e. a_{ij} represents the distance between location i and j , and $\mathbf{B}$ is normally referred to as the flow matrix, i.e. b_{ij} denotes the flow of materials from facility i to j . The goal is to find a permutation π of the set $\mathbf{M} = \{1, 2, 3, \dots, n\}$, to minimize the objective function $C(\pi)$ described by:

$$C(\pi) = \sum_{l=1}^n \sum_{t=1}^n a_{lt} b_{\pi(l)\pi(t)} \quad \text{Eq. 8-1}$$

In the above equation, $\pi(l)$ and $\pi(t)$ denote the l -th and the t -th element respectively in the permutation π .

There are many applications that can be modeled as QAP. Besides the facility location problem, other applications of QAP include backboard wiring [104], campus planning [105], ranking of archaeological data [106] and hospital layout planning [107], numerical analysis [108], placement of electronic components [109-112] and so on. More applications of QAP can be found in [113].

The QAP has been a topic of frequent study of combinatorial optimization not only due to its applicable and theoretical importance but its computational complexity. Since the QAP is NP-hard, only implicit enumeration approaches are known to solve them optimally. In recent years, some important real-life and benchmark problems in QAPLIB [114] have been solved to optimality by some branch-and-bound variations. For example, *Ste36b-c* were solved by Nystrom in 1999; *Bur26a* by Hahn in 2001; *Ste36a* by Brixius and Anstreicher [115]; *Kra30b*, *Kra32* and *Tho30* by Anstreicher et al. in 2000; *Kra30a* by Hahn [116, 117]; *Tai25a* by Hahn in 2003 and *Bur26b-h* by Hahn .

However, the computational time cost will increase exponentially with respect to n and thus become unmanageable. Generally, it is considered unsolvable for QAP with size larger than 30 through exhaustive methods. But in practice, many applications result in instances that are of significantly larger dimension than 30.

Inevitably, heuristic, and more recently, meta-heuristic approaches have played important role in algorithms capable of providing good solutions in reasonable time. In the last decade, much research has been devoted into the development of this class of approaches for solving QAP.

Genetic algorithms (GAs) refer to the class of algorithms that simulates the natural evolutionary process. Through the genetic operators like selection, crossover and mutation, it evolves a population of candidate individuals (solutions) and converges to the global optimum in an iterative manner [24, 118]. Applications of GA on QAP include Brown et al. [119], Bui and Moon [120], Tate and Smith [121], Mavridou and Pardalos [122], Merz [123, 124], Kochhar et al. [125], Tavakkoli-Moghaddain and Shayan [126], Gong et al. [127], Lim et al. [39, 40], Drezner [43, 128, 129], El-Baz [130], Wang and Okazaki [131] and so on. In [42], Drezner proposed a two-phase genetic algorithm. The first phase is a quick genetic strategy that is used to produce the initial solutions for the second phase. This compounded GA produced very good solutions with manageable time complexity for the symmetric class of QAP benchmark problems.

Ant colony optimization (ACO) refers to the class of algorithms that imitate real ant colony's social behaviors in finding a food source. As introduced in Section 3.2, ACO has been frequently applied to solve various combinatorial optimization problems including QAP. Numerical results of ACO for the QAP can be found in Maniezzo and Colormi [72, 132], Colormi et al. [73], Dorigo et al. [47], Gambardella et al. [74], Stuetzle and Dorigo [50], Stuetzle and Holger [56], Talbi et al. [75], Middendorf et al. [63], Solimanpur et al. [76], Randall [77] and Ying and Liao [78].

Local Search refers to the class of heuristic algorithms that start with a single point in the solution space and search in its “neighbourhood” by localized movement. Examples of local search include simulated annealing and tabu search.

Simulated annealing [70] is a local search algorithm that exploits the analogy between combinatorial optimization algorithms and statistical mechanics. The acceptance or rejection of the neighbourhood movement is controlled by the “states energy”. The states energy are updated (increase or decrease) as the search progresses in order to escape from the local optima. Burkard and Rendl [133] proposed the first application of simulated annealing to the QAP. Other examples include Wilhelm and Ward [134], Connolly [135], Laursen [136], Abreu et al. [137], Bos [138], Yip and Pao [139], Burkard and Cela [140], Peng et al. [141],

Tian et al. [142], Chiang and Chiang [143], Misevicius [144, 145], Tsuchiya et al. [146], Siu and Chang [147] and Baykasoglu [148].

Tabu search is a local search algorithm introduced by Glover (1989a,b) to find good quality solutions for integer programming problems. It has a memory unit, namely tabu list, to record the paths it has searched and thus avoid searching the same place redundantly. The acceptance or rejection of a neighbourhood movement is controlled by the criteria which are normally called tabu rules. Applications of tabu search for the QAP can be found in Skorin-Kapov [149, 150], Taillard [151], Bland and Dawson [152], Rogger et al. [153], Chakrapani and Skorin-Kapov [154], Misevicius [144, 155] and Drezner [128]. Furthermore, a comparison between various tabu search approaches and the genetic algorithm, when applied to QAP, can be found in Taillard [79].

Other heuristic algorithm applications to QAP include greedy randomized adaptive search procedure [71], neural networks [156], scatter search [157], variable neighborhood search (VNS) [158], iterated local search [41] and so on. More algorithms for QAP can be found in [113].

8.2 Automatic Configuration Engine Setup for QAP

In my work, I configured an experiment within ADEP to validate the effectiveness of the automatic configuration engine when applied to the QAP. The setup of the automatic configuration engine as described in Chapter 7 is presented as follows.

8.2.1 Algorithm evaluation

Before starting the training process of the automatic configuration engine, it is necessary to consider how users can evaluate the performance of an algorithm. This has to do with the domain specific nature of the problem(s) being addressed, the user requirements and expectations on the algorithm according to various practical scenarios and probably many other subjective as well as objective factors.

In my work on solving QAP, I take two performance metrics to measure an algorithm, average error (*avg_gap*) and average computational time before termination of each trial (*T*). In particular, *avg_gap* refers to the difference between the average objective value of the solutions obtained for all the trial runs and the best-known value of the objective function. It serves as a performance metric for measuring the effectiveness of an algorithm. On the other hand, *T* indicates the average computational time for all the trial runs before the termination of the program. It is an indicative measure of the efficiency of an algorithm.

It is worth noting that, besides the *avg_gap* and *T*, there are many other possible performance metrics. For example, to measure the stability of an algorithm, user may use the standard deviation measure of the solutions obtained or the success rate (R_s), which refers to the ratio of number of trials the algorithm finds the best-known solution to the total number of trials.

I associate a fitness value with an algorithm, which is used to indicate how good the algorithm is. Based on the performance metrics described above, users may define the fitness value for every candidate algorithm according to their requirements. In my experiment, I explored two forms of fitness function (criteria) for the algorithm to test the robustness of the automatic configuration engine to suit user's requirements.

The first form of fitness function (*algo_FF1*) is defined as:

$$f(algo) = \frac{1}{avg_gap + \varepsilon} \quad \text{Eq. 8-2}$$

where $f(algo)$ stands for the fitness value of the algorithm and ε is a very small real number, which is used to avoid invalid division operation (when *avg_gap* is zero).

CHAPTER 8. Validation on Automatic Configuration

It is clear that the first fitness function (algo_FF1) places all the priority on effectiveness, i.e. the solution quality, and does not consider computational time to be important.

On the other hand, in many practical scenarios, users may have concerns on both effectiveness (solution quality) and efficiency of the algorithm. Therefore I designed and tested the second form of fitness function referred to as algo_FF2. It attempts to achieve a balance between effectiveness and efficiency:

$$f(algo) = \frac{1}{avg_gap \times T + \varepsilon} \quad \text{Eq. 8-3}$$

The equation above (algo_FF2) takes both solution quality and the computational time into account.

There are various other possible ways to design the fitness function for an algorithm to evaluate its performance. Even for the algo_FF2, there could be many variations, e.g., assign and tune the weights for *avg_gap* and *T* respectively to achieve more desirable form of balance between effectiveness and efficiency.

8.2.2 Trace update and initialization

With the algorithm evaluation mechanism described earlier, users can sort the population of algorithms (descending order in fitness value) in each iteration of the training phase as illustrated earlier in Figure 7-1. I define $AlgorithmPop[j]$ ($j=0,1,\dots,popsize$) as the j -th algorithm in the population after sorting. Though there are many possible ways to update the trace information based on the generated algorithms in each iteration, in my work, I update the trace information as follows:

$$\tau_{(i+1)}^k = \begin{cases} \tau_{(i)}^k + \left(\frac{1}{2}\right)^j & \text{if } AlgorithmPop[j] \text{ covers } \tau_{(i)}^k \\ \text{remains} & \text{else} \end{cases} \quad \text{Eq. 8-4}$$

where $\tau_{(i)}^k$ indicates the trace value of the k -th branch of a traversal split node in iteration i . An algorithm will lay a certain amount of trace value on the root-leaf paths that it covers. From the above equation, it is clear that the better the algorithm is, the more trace value it lays on the paths it covers. For example, the best algorithm of a given iteration lays “1.0” on each branch that it covers and the second best lays “0.5” on what it covers. In implementation, I only allow the top 3 algorithms of each iteration to lay the trace information on their paths. This way, the good features and configurations are emphasized in a cumulative manner and the weak ones are deemphasized accordingly. Therefore in a probabilistic way, the

training/learning process converges to a stable state (e.g. some branches are dominating or the best found algorithm dose not change for a frame of time). This state gives us the optimal or very good configurations of the algorithm for the problem being addressed.

Given the trace update mechanism described above, I simply initialize the trace values by a constant Δ . It is worth noting that the value of Δ may influence the convergence of the training process. The larger Δ is, the slower the convergence is, and vice versa. After a series of experiments, the value of Δ was fine-tuned to be 20.

8.3 Simulation Results and Analysis

Considering the complexity of the QAP, I chose small but non-trivial instances from QAPLIB [114] as the training set and large instances as the testing set. It randomly chooses a QAP instance from the training set and used it to train the tree structures for 2 iterations and then chooses another training instance. I set the algorithm population size to 10 and the maximum number of iterations to 200. During the training phase, each algorithm undergoes 10 trials to derive average performance and during the testing phase, 20 trials were carried out. Each trial is terminated if there is no improvement in the latest successive $20 \times \text{population_size}$ number of generated solutions. I restricted the learning to the genetic algorithm

module although ACO module was still taken into account as one option for the population initialization procedure.

8.3.1 Simulation results

The experiments using both algo_FF1 and algo_FF2 were carried out for several trials. It is observed that every trial of training obtained either exactly the same or very similar algorithms. Figure 8-1 shows the two resulting algorithms through training using algo_FF1 and algo_FF2.

Figure 8-1. Resulting hybrid genetic algorithms

CHAPTER 8. Validation on Automatic Configuration

It is observed that the general structures of the two algorithms are very similar. However, the differences in the configurations of the two algorithms, which have been highlighted in the figure with bold rectangular blocks, make them perform quite differently. Again the ADEP was compiled using Visual C++ 6.0 and run on an NEC 2.4 GHz PC with 512 MB memory. The numerical results of the simulation are summarized in

Table 8-1. Simulation results on QAP benchmark problems

problem name	problem size	algo_FF1			algo_FF2		
		avg_gap	T (s)	Rs	avg_gap	T (s)	Rs
Training							
chr25a	25	0.000%	13.38	100%	0.000%	12.57	100%
kra30a	30	0.000%	6.24	100%	0.000%	6.12	100%
nug30	30	0.000%	11.93	100%	0.000%	5.93	100%
kra30b	30	0.000%	13.87	100%	0.000%	10.56	100%
tho40	40	0.010%	52.78	30%	0.032%	9.53	10%
tai40a	40	0.328%	88.40	0%	0.592%	47.80	0%
tai40b	40	0.000%	4.75	100%	0.000%	3.927	100%
lipa40a	40	0.000%	15.73	100%	0.000%	13.020	100%
sko42	42	0.000%	30.43	100%	0.000%	28.845	100%
wil50	50	0.002%	229.75	90%	0.003%	102.561	70%
tai50a	50	0.533%	174.70	0%	0.627%	120.69	0%
Testing							
tai60a	60	0.598%	338.3	0%	0.683%	185.68	0%
sko72	72	0.000%	530.8	100%	0.017%	276.39	45%
tai80a	80	0.416%	833.0	0%	0.769%	399.589	0%
sko81	81	0.008%	747.0	80%	0.017%	348.38	20%
sko90	90	0.005%	1043.8	45%	0.012%	528.53	20%
sko100a	100	0.007%	1495.9	75%	0.014%	796.36	30%
sko100b	100	0.000%	1534.6	100%	0.003%	739.69	75%
sko100c	100	0.000%	1486.3	100%	0.005%	753.87	60%
sko100d	100	0.007%	1489.4	70%	0.017%	803.74	30%
sko100e	100	0.000%	1496.7	100%	0.002%	716.430	80%
sko100f	100	0.012%	1567.5	35%	0.020%	815.83	30%
wil100	100	0.002%	1486.4	35%	0.004%	719.59	20%
tai100a	100	0.581%	1543.9	0%	0.672%	849.56	0%
tho150	150	0.056%	2869.4	0%	0.078%	1639.58	0%

From the table, it is shown that through the training process based on algo_FF1, the automatic configuration engine is able to produce an algorithm (hybrid genetic algorithm) that generates very good solutions in both training and testing phases. It is able to find the best-known solutions for 20 out of 25 instances. It is also observed that, training through algo_FF2 results in an efficient algorithm as evidenced by the fact that the computational time is significantly less than that of algo_FF1 for all instances tested. However, the solution quality with algo_FF2 shows no significant compromise compared to that of algo_FF1. With algo_FF2, the automatic configuration engine is able to produce an algorithm which achieves a good balance between the two performance metrics. This feature can be very useful, because in practical scenarios, users may have very different requirements and expectations on the algorithm. My experiment shows that the automatic configuration engine is able to learn based on user requirements and in turn produce an algorithm which satisfies the user's needs.

I compared the resulting hybrid genetic algorithm using algo_FF1 with two other state-of-the-art algorithms recently reported in the literature for solving QAP, namely, the compounded GA of Drezner [42] and the population-based iterated local search (ILS) of Stuetzle [41]. The comparison is summarized in Table 8-2.

CHAPTER 8. Validation on Automatic Configuration

Table 8-2. Comparison with two other algorithms for QAP

problem name	problem size	Compounded GA				Population-based ILS				Resulting hybrid-GA		
		avg_gap	$T(s)^a$	Rs		avg_gap	$T(s)^b$	Rs		avg_gap	$T(s)^c$	Rs
Symmetic problems												
sko72	72	0.000%	718.8	100%		0.0012%	1200.0	88%		0.000%	530.8	100%
sko81	81	0.002%	1188.0	80%		0.0074%	1200.0	52%		0.008%	747.0	80%
sko90	90	0.001%	1810.2	85%		0.0057%	1200.0	40%		0.005%	1043.8	45%
sko100a	100	0.003%	2799.6	85%		0.0120%	1200.0	30%		0.007%	1495.9	75%
sko100b	100	0.000%	2903.4	100%		0.0068%	1200.0	50%		0.000%	1534.6	100%
sko100c	100	0.000%	2738.4	100%		0.0023%	1200.0	60%		0.000%	1486.3	100%
sko100d	100	0.005%	2704.2	85%		0.0210%	1200.0	0%		0.007%	1489.4	70%
sko100e	100	0.000%	2745.6	100%		0.0013%	1200.0	70%		0.000%	1496.7	100%
sko100f	100	0.007%	2843.4	40%		0.0370%	1200.0	0%		0.012%	1567.5	35%
wil100	100	0.003%	2751.6	45%		0.0041%	1200.0	10%		0.002%	1486.4	35%
tho150	150	N/A	N/A	N/A		0.0680%	3600.0	0%		0.056%	2869.4	0%
Asymmetic probimes												
tai40a	40	N/A	N/A	N/A		0.28%	1200.0	0%		0.328%	88.4	0%
tai50a	50	N/A	N/A	N/A		0.61%	1200.0	0%		0.533%	174.7	0%
tai60a	60	N/A	N/A	N/A		0.82%	1200.0	0%		0.598%	338.3	0%
tai80a	80	N/A	N/A	N/A		0.62%	1200.0	0%		0.416%	833.0	0%
tai100a	100	N/A	N/A	N/A		0.69%	1200.0	0%		0.581%	1543.9	0%

^a Computational time on a 2.8G Hz computer^b Computational time on a 1.3G Hz computer^c Computational time on an NEC 2.4G Hz computer

It is observed from the table that the resulting algorithm is more superior compared to the population-based ILS algorithm proposed in [41]. This is evident from the fact that the hybrid genetic algorithm produces significantly better average solutions for all the instances and higher success rate for most of the instances being considered. Comparison of hybrid genetic algorithm to the compounded GA [42] for the symmetric QAP problems further highlights the superiority of my approach by generating competitive solution qualities at significantly lower computational cost on the same QAP problems. Considering the fact that my algorithm produced the best (or equal) average solutions (highlighted in bold) in 10

out of total 16 problems being tested, I contend that the hybrid genetic algorithm developed through training is one of the state-of-art algorithms for solving QAP.

It is very important to note that the two algorithms being compared with are designed by very experienced experts in both meta-heuristic algorithms as well as in QAP. The local search procedures used in the two algorithms (the extended concentric tabu search in [42] and the iterated local search in [41]) are designed and proposed by two experienced researchers recently. In contrast, the ADEP library contains only very basic and well-known procedures. Still the resulting algorithm produced human-competitive or even better performance. I contend that it is the automatic configuration engine that tunes the algorithm to its full potential in achieving superior performance.

8.3.2 Trace analysis

I analyzed the trace information as the training progresses in order to understand how and why the automatic configuration engine works. It is also very helpful for the researchers in studying various meta-heuristic algorithms.

Figure 8-2 shows how the *Population Initialization* trace values change as the training progresses. It is shown that, for both algo_FF1 and algo_FF2, random initialization is more superior over ACO and GRASP procedures. This can be explained by the well-known theory that in GA, the initial population should be as

CHAPTER 8. Validation on Automatic Configuration

diverse as possible in order to uncover global optimum. There are many ACO-GA hybrid algorithms proposed recently. Using ADEP, it is informally proven that using ACO or GRASP as the initial solution generator is not a good idea.

The superiority of random initialization is even more obvious in algo_FF2. This can be explained by the fact that both ACO and GRASP are much more time-consuming than random generation. Considering the objective of algo_FF2, it is not surprising to find such overwhelming superiority of random generation in algo_FF2.

a)

b)

Figure 8-2. Trace Analysis - Population Initialization

However, some trace values change in very different ways as the training progresses. Figure 8-3 shows how the Population Size trace value changes in algo_FF1 and algo_FF2 respectively.

a)

b)

Figure 8-3. Trace Analysis – Population Size

It is shown that in algo_FF1, which does not take the computational time into account, larger population size seems advantageous. This is reasonable because in GA, large population size normally leads to better solution quality at a price of more computational time. However in algo_FF2, which is designed to achieve a balance between effectiveness and efficiency, large population size may not be a good choice any more due to its expensive computational time. Instead, a medium size population seems to be a better option.

The tendency of how the traces develop is consistent with people's intuition. This shows that the automatic configuration engine trains the tree structure in a very reasonable and intelligent way. It learns not only from the domain specific nature of the problems but also the user requirements and expectations and then converges towards the appropriate paths of the tree. There are many more similar analyses that can be done through ADEP. These can serve as one avenue for future exploration. I believe this kind of analyses may be helpful for researchers to investigate the hidden features of various meta-heuristic procedures and thus to guide and inspire them to design more powerful algorithms in the future.

Chapter 9 Conclusions and Future Directions

9.1 Conclusions

Conventional algorithm development is time consuming and tedious, especially in the part on generation and modification of codes. It cannot catch up with the increasing complexity of problems as well as the algorithm solver. My first stage of work addresses the need of quick design and generation of efficient meta-heuristic algorithms for solving a class of combinatorial optimization problems. The paradigm of the novel algorithm development methodology was implemented through an integrated problem-solving environment, namely ADEP. ADEP incorporates a comprehensive procedure library, which is managed by a novel data structure, namely left variation – right property tree structure. With this data structure, the ADEP library is scalable and portable, open for incorporation of new procedural elements. Users may configure a meta-heuristic algorithm through an intuitive graphical user interface in ADEP. The algorithm configured can be generated automatically as a stand-alone program. With the elegant design of the data management mechanism, i.e. the LVRP trees structure, only minimal effort is required for customizing the generated program to solve a particular class of

problems. This way, the most tedious part in conventional algorithm development is avoided. Therefore the efficiency of the algorithm development is enhanced significantly. I carried out case studies based on benchmark problems and problems in real-life applications. It is shown that, using the proposed methodology in ADEP, the algorithm development cycle is reduced significantly. Although the extent of the reduction in time is extremely hard to quantify, based on my own experience, the time for fastest prototyping of a meta-heuristic algorithm (without learning) can be reduced to hours. However, it is necessary to mention that this statement is definitely case-dependent.

Although a significant part of the algorithm can be generated automatically, the quality of the algorithm still depends very much on the user's experience, domain specific knowledge on the problems and the algorithm design skill. Furthermore, in many practical scenarios, users may impose various requirements on the algorithm, which may make an algorithm work well in one scenario, while failing miserably in another. In the second stage of work, I proposed a novel evolutionary learning methodology based on the novel LVRP tree structure, which serves as the automatic configuration engine in ADEP. It is able to automatically learn from both the specific domain features of the problem and the user requirements on the algorithm, and then optimize the configurations of an algorithm for solving the class of problems being addressed. I tested the automatic configuration engine on various instances of the QAP. The simulation results show that the automatic

configuration engine is able to produce a powerful algorithm that performs even better than recently reported algorithms in the literature. Furthermore, the automatic configuration engine is also able to produce another algorithm which is trained based on a different set of user requirements. This feature is useful because in practical scenarios, it is very normal to impose performance requirements that are unique or even dynamic in nature.

9.2 Future Research

Although the proposed algorithm development methodology was shown to be effective and useful in solving both benchmark problems and real-life problems, there is still a lot of room for future research.

9.2.1 Short term

The most straightforward short term extension of the work is to make the ADEP library more comprehensive. Considering the scalable structure of the LVRP tree structure, this work should be relatively straightforward.

Another short term extension can be the development of a friendlier user interface to further facilitate the user in configuring metaheuristics. A further extension based on this is to develop the interface for the user to define the problem-domain

information and formulate the objective function in a high level manner, instead of dealing with source codes. Ideally, the code generator is able to generate objective function formulated by the user through some very high level specifications. This will enhance the efficiency and the productivity of algorithm development dramatically.

Local search is a big class of metaheuristics, with diverse applications. The development of a local search module will be a great improvement to the applicability of ADEP. Other metaheuristics like particle swarm optimization (PSO) will serve as further improvement to ADEP.

9.2.2 Long term

The automatic configuration engine has one drawback. It cannot change the topological structure of the tree once the training starts. On the other hand, genetic programming technique has no restriction on a tree's topological structure. The most exciting long-term future research direction can be the exploration of the possibility to combine genetic programming and the automatic configuration engine, taking advantage of both methodologies. The objective is to enable the automatic configuration engine to explore a much greater algorithm configuration space.

CHAPTER 9. Conclusions and Future Directions

Exploration of other algorithms, rather than just metaheuristics, will be another exciting research direction. Also, extension from permutation-based solution representation to more general solution representations will be another great improvement over existing work.

Chapter 10 Author's Publications

Patent

[P1]Method and Apparatus for an Algorithm Development Environment for Solving a Class of Real-Life Combinatorial Optimisation Problems. PCT Patent Application No. PCT/SG2006/000382, filed on Dec. 8, 2006. (Inventors: M.H. Lim and Y.L. Xu).

[P2]Method and Apparatus for Automatic Configuration of Meta-heuristic Algorithms in a Problem Solving Environment. PCT Patent Application No. PCT/SG2006/000403, filed on Dec. 22, 2006. (Inventors: M.H. Lim , Y.L. Xu and Y.S. Ong).

Journal

[J1] M. H. Lim and Y. L. Xu. Application of Hybrid Genetic Algorithm in Supply Chain Management. International Journal of Computers, Systems and Signals, Vol. 6(1), 2005.

- [J2] Y. L. Xu, M. H. Lim and Y. S. Ong. A Tabu-Memetic Algorithm for Solving Quadratic Assignment Problems. In submission to Operations Research Letters.

Conference

- [C1] Y. L. Xu, M. H. Lim, Y. S. Ong and J. Tang. A GA-ACO-Local Search Hybrid Algorithm for Solving Quadratic Assignment Problem. Genetic and Evolutionary Computation Conference (GECCO), Seattle, 2006.
- [C2] Y. L. Xu, M. H. Lim and M. J. Er. Investigation on Genetic Representations for Vehicle Routing Problem, IEEE International Conference on System, Man and Cybernetics, Hawaii, 2005.
- [C3] Y. L. Xu and M. H. Lim. Solving constraint satisfaction optimization problem using an integrated algorithm development environment, In Proceedings of 10th international symposium on integrated circuits, Singapore, 2004.
- [C4] A. Agarwal, M. H. Lim, Y. L. Xu and Y. S. Ong. Evolutionary Graph Mining for the Discovery of Site Visitation Sequences for a Single URAV. The second International Conference on Computational Intelligence, Robotics and Autonomous Systems, Singapore, 2003.

[C5] A. Agarwal, M. H. Lim and Y. L. Xu. Graph Compression via Compaction of Degree-2 Nodes for URAV Visitation Sequencing. Unmanned Vehicle System Technologies, Royal Military Academy , Brussels, 2003.

[C6] Y. L. Xu, M. H. Lim and K. H. Yan. Bias suppression with stochastic selection local search in hybrid-GA for solving quadratic assignment problems. The second International Conference on Computational Intelligence, Robotics and Autonomous Systems, Singapore, 2003.

Bibliography

- [1] T. C. Koopmans and M. J. Beckmann, "Assignment problems and the location of economic activities," *Econometrica*, pp. 53-76, vol. 25, 1957.
- [2] M. M. Flood, "The Traveling Salesman Problem," *Operations Research*, pp. 61-75, vol. 4, 1956.
- [3] G. Culler and B. Fried, "An on-line computing center for scientific problems," *Proc. of IEEE Pacific Computer Conference*, pp. 221, March, 1963.
- [4] J. Rice and S. Rosen, "Napss - a numerical analysis problem solving system," *Proc. of ACM National Conference*, ACM Press, pp. 51-56, 1966.
- [5] M. Klerer and J. Reinfelds, *Interactive System for Experimental Applied Mathematics*: Academic Press, 1968.
- [6] E. N. Houstis and J. R. Rice, "On the Future of Problem Solving Environment," *Internal Technical Report, Department of Computer Science, Purdue University*, 2000.
- [7] Y. Yoshida and N. Adachi, "System, method and program storage medium for executing a learning algorithm." U.S. Pat. No. 6,263,325 B1, 2001.
- [8] "Epogy Process Integration and Design Optimization Software. Download at: http://www.synaps-inc.com/PDF/Epogy_Brochure.pdf."
- [9] S. Waldon, D. Powell, and S. Tong, "User directed heuristic design optimization search." U.S. Pat. No. 6,086,617, 2000.

- [10] A. S. Egilsson, "Graphical environment for managing and developing applications." U.S. Pat. No. 6,286,017, 2001.
- [11] P. Collet, E. Lutton, M. Schoenauer, and J. Louchet, "Take it EASEA," *Proc. of 6th International Conference on Parallel Problem Solving from Nature*, pp. 891-901, Paris, France, September, 2000.
- [12] C. Veenhuis, K. Franke, and M. Koppen, "A semantic model for evolutionary computation," *Proc. of 6th International Conference on Soft Computing*, Fukuoka, Japan, 2000.
- [13] A. Fink, S. Voss, and D. L. Woodruff, "Metaheuristic Class Libraries," in *Handbook of Metaheuristics*, vol. 57, F. Glover and G. A. Kochenberger, Eds. New York: Springer, 2003, pp. 515-535.
- [14] M. Birrattari, T. Stutezle., L. Paquete, and K. Varrentrapp, "A Racing Algorithm for Configuring Metaheuristics," *Proc. of Genetic and Evolutionary Computation Conference (GECCO'02)*, New York, USA, July, 2002.
- [15] B. Adenso-Diaz and M. Laguna, "Fine-Tuning of Algrorithms using Fractional Experimental Designs and Local Search," *Operations Research*, vol. 54, pp. 99-114, 2006.
- [16] M. Birattari, "The Problem of Tuning Metaheuristics, as Seen from a Machine Learning Perspective." *Ph.D. Thesis*, Brussels, Belgium: Universite Libre de Bruxelles, 2004.

- [17] M. Oltean, "Evolving Evolutionary Algorithms Using Linear Genetic Programming," *Evolutionary Computation*, vol. 13(3), 2005.
- [18] K. O. Stanley, B. D. Bryant, and R. Miikkulain, "Real-Time Neuroevolution in the NERO Video Game," *IEEE Transactions on Evolutionary Computation*, pp. 653-668, vol. 9(6), 2005.
- [19] J. R. Koza, I. Bennett, H. Forrest, Andre, David, Keane, and A. Martin, "Method and apparatus for automated design of complex structures using genetic programming." U.S. Pat. No. 6,360,191: J. R. Koza (Los Altos Hills, CA), 2002.
- [20] J. R. Koza, I. Bennett, H. Forrest, A. David, and M. A. Keane, "Genetic programming problem solver with automatically defined stores loops and recursions." U.S. Pat. No. 6,532,453: J. R. Koza (Los Altos Hills, CA), 2003.
- [21] P. W. Worzel, "Method and system for genetic programming." U.S. Pat. No. 6,327,582, 2001.
- [22] J. H. Holland, *Adaptation in Natural and Artificial Systems*: MIT Press, 1975.
- [23] C. R. Reeves and J. E. Rowe, *Genetic Algorithms: Principles and Perspectives*. Norwell, MA: Kluwer, 2001.
- [24] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Wokingham, England: Addison-Wesley, 1989.

- [25] J. Rees and G. J. Koehler, "A investigation of GA performance results for different cardinality alphabets.," in *Evolutionary Algorithms*, vol. 111, L. D. Davis, K. DeJong, M. D. Vose, and L. D. Whitley, Eds. New York: Springer-Verlag, 1999, pp. 191-206.
- [26] M. D. Vose, "Modeling simple genetic algorithms.," in *Foundations of Genetic Algorithms 2*, L. D. Whitley, Ed. San Mateo, CA: Morgan Kaufmann, 1993.
- [27] D. E. Goldberg, "Sizing Populations for serial and parallel genetic algorithms," *Proc. of the 3rd International Conference on Genetic Algorithms*, San Mateo, CA, June, 1989.
- [28] D. E. Goldberg, "Optimal initial population size for binary-coded genetic algorithms," University of Alabama, Tuscaloosa, TAGA Report 85001, 1985.
- [29] J. J. Grefenstette, "Optimization of control parameters for genetic algorithms," *IEEE Trans. on SMC*, vol. 16, pp. 122-128, 1986.
- [30] J. D. Schaffer, R. A. Caruana, L. J. Eshelman, and R. Das, "A study of control parameters affecting online performance of genetic algorithms for function optimization," *Proc. of the 3rd International Conference on Genetic Algorithm*, Los Altos, CA, June, 1989.
- [31] C. R. Reeves, "Using genetic algorithms with small populations," *Proc. of the 5th International Conference on Genetic Algorithms*, Urbana-Champaign, IL, USA, June 1993.

- [32] P. J. B. Hancock, "An empirical comparison of selection methods in evolutionary algorithms," in *Evolutionary Computing: AISB Workshop*, T. C. Fogarty, Ed. Berlin: Springer-Verlag, 1994.
- [33] J. E. Baker, "Reducing bias and inefficiency in the selection algorithm," *Proc. of 2nd International Conference on Genetic Algorithm*, Cambridge, MA, USA, July 1987.
- [34] R. Allenson, "Genetic Algorithms with Gender for Multi-Function Optimisation," *Technical Report of Edinburgh Parallel Computing Centre*, vol. EPCC-SS92-01, 1992.
- [35] J. Sanchez-Velazco and J. A. Bullinaria, "Gendered Selection Strategies in Genetic Algorithms for Optimization," *Proc. of the UK Workshop on Computational Intelligence (UKCI 2003)*, Bristol, UK, September, 2003.
- [36] Z. Drezner and T. Drezner, "Gender Specific genetic Algorithms," *Information Systems and Operations Research*, vol. 44, pp. 117-127, 2006.
- [37] K. A. D. Jong, "An analysis of the behavior of a class of genetic adaptive systems.," *Doctoral dissertation of University of Michigan*, 1975.
- [38] D. E. Goldberg and K. Deb, "A comparative analysis of selection schemes used in genetic algorithms.," in *Foundations of Genetic Algorithms*, G. J. E. Rawlins, Ed. San Mateo, CA: Morgan Kaufmann, 1991.
- [39] M. H. Lim, Y. Yuan, and S. Omatu, "Efficient genetic algorithms using simple genes exchange local search policy for the quadratic assignment

- problem," *Computational Optimization and Applications*, vol. 15, pp. 249-268, 2000.
- [40] M. H. Lim, Y. Yuan, and S. Omatu, "Extensive testing of a hybrid genetic algorithm for quadratic assignment problem," *Computational Optimization and Applications*, vol. 23, pp. 47-64, 2002.
- [41] T. Stuetzle, "Iterated local search for the quadratic assignment problem," *European Journal of Operational Research*, in press.
- [42] Z. Drezner, "Compounded genetic algorithms for the quadratic assignment problem," *Operations Research Letters*, vol. 33, pp. 475-480, 2005.
- [43] Z. Drezner, "A new genetic algorithm for the quadratic assignment problem," *Infors Journal on Computing*, vol. 15, pp. 320-330, 2003.
- [44] Y. S. Ong, M. H. Lim, N. Zhu, and K. W. Wong, "Classification of Adaptive Memetic Algorithms: A Comparative Study," *IEEE Transactions on Systems, Man and Cybernetics - Part B*, vol. 36, pp. 141-152, 2006.
- [45] M. Dorigo, "Optimization, Learning and Natural Algorithms," Ph.D. Thesis, Department of Systems and Information Electronic Engineering, Politecnico di Milano, Italy, 1992.
- [46] M. Dorigo and L. M. Gambardella, "Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, pp. 53-66, 1997.

- [47] M. Dorigo, V. Maniezzo, and A. Colorni, "The ant system: Optimization by a colony of cooperating agents," *IEEE Transaction on Systems, Man, and Cybernetics - Part B*, vol. 26, pp. 29-42, 1996.
- [48] E. Taillard and L. M. Gambardella, "An Ant Approach for Structured Quadratic Assignment Problems," *Proc. of the 2nd Metaheuristics International Conference*, Sophia-Antipolis, France, July, 1997.
- [49] L. M. Gambardella, E. Taillard, and M. Dorigo, "Ant Colonies for the Quadratic Assignment Problem," *Journal of the Operational Research Society*, vol. 50, pp. 167-176, 1999.
- [50] T. Stuetzle and M. Dorigo, "ACO algorithms for the quadratic assignment problem," in *New Ideas in Optimization*, D. Corne, M. Dorigo, and F. Glover, Eds.: McGraw-Hill, pp. 33-55, 1999.
- [51] Y. Tian, J. Song, D. Yao, and J. Hu, "Dynamic vehicle routing problem using hybrid ant system," *Proc. of the 2003 IEEE International Conference on Intelligent Transportation Systems*, vol. 2, pp. 970-974, 2003.
- [52] M. Reimann, K. Doerner, and R. F. Hartl, "D-ants: savings based ants divide and conquer the vehicle routing problem," *Computers & Operations Research*, vol. 31, pp. 563-591, 2004.
- [53] L. M. Gambardella, E. Taillard, and G. Agazzi, "MACS-VRPTW: A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows," in *New Ideas in Optimization*, D. Corne, M. Dorigo, and F. Glover, Eds., pp. 63-76, 1999.

- [54] B. Bullnheimer, R. F. Hartl, and C. Strauss, "Applying the Ant System to the Vehicle Routing Problem," presented at 2nd International Conference on Metaheuristics, Sophia-Antipolis, France, July, 1997.
- [55] M. Dorigo, V. Maniezzo, and A. Coloni, "Ant System: Optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, vol. 26, pp. 29-41, 1996.
- [56] T. Stuetzle and H. H. 2000, "MAX-MIN ant system," *Future Generation Computer Systems*, vol. 16, pp. 889-914, 2000.
- [57] E. D. Taillard and L. M. Gambardella, "Adaptive Memories for the Quadratic Assignment Problem," *Technical Report IDSIA-87-97, Lugano, Switzerland*, 1997.
- [58] V. Maniezzo, "Exact and Approximate Nondeterministic Tree-Search Procedures for the Quadratic Assignment Problem," *Technical Report CSR 98-1, University of Bologna*, 1998.
- [59] T. Stuetzle and H. Hoos, "The MAX-MIN Ant System and Local Search for the Traveling Salesman Problem," *Proc. of the IEEE International Conference on Evolutionary Computation*, pp. 308-313. Indianapolis, Indiana, USA, April, 1997.
- [60] M. Birrattari, T. Stuetzle, L. Paquete, and K. Varrentrapp, "A Racing Algorithm For Configuring Metaheuristics," *Proc. of the Genetic and Evolutionary Computation Conference*, pp. 11-18, New York, July, 2002.

- [61] M. Reimann, S. Shtovba, and E. Nepomuceno, "A hybrid ACO-GA approach to solve Vehicle Routing Problems," *Student Papers of the Complex Systems Summer School, Budapest, Santa Fe Institute*, 2001.
- [62] A. Acan, "GAACO: A GA + ACO Hybrid for Faster and Better Search Capability." *Proc. of the 3rd International Workshop on Ant Algorithms (ANTS'02)*, pp. 300-301. Brussels, Belgium, 2002.
- [63] M. Middendorf, F. Reischle, and H. Schmeck, "Multi colony ant algorithms," *Journal of Heuristics*, vol. 8, pp. 305-320, 2002.
- [64] F. Glover, "Tabu search - Part II," *ORSA Journal on Computing*, vol. 2, pp. 4-32, 1989.
- [65] F. Glover, "Tabu search - Part I," *ORSA Journal on Computing*, vol. 1, pp. 190-206, 1989.
- [66] E. D. Taillard, "Robust taboo search for the quadratic assignment problem," *Parallel Computing*, pp. 443-455, vol. 17, 1991.
- [67] E. D. Taillard, "Comparison of iterative searches for the quadratic assignment problem," *Location science*, vol. 3, pp. 87-105, 1995.
- [68] R. Battiti and G. Tecchiolli, "Teh reactive tabu search," *ORSA Journal on Computing*, vol. 6, pp. 126-140, 1994.
- [69] H. R. Lourenco, O. Martin, and T. Stuetzle, "Iterated Local Search," in *Handbook of Metaheuristics*, F. Glover and G. Kochenberger, Eds. Norwell, MA: Kluwer Academic Publishers, pp. 321-353, 2002.

- [70] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671-680, 1983.
- [71] Y. Li, P. M. Pardalos, and M. Resende, "A greedy randomized adaptive search procedure for the quadratic assignment problem," in *Quadratic Assignment and Related Problems*, vol. 16, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, P. Pardalos and H. Wolkowicz, Eds. Rhode Island: AMS: Providence, pp. 173-187, 1994.
- [72] V. Maniezzo and A. C. 1999, "The ant system applied to the quadratic assignment problem," *Knowledge and Data Engineering*, vol. 11, pp. 769-778, 1999.
- [73] A. Colomi, M. Dorigo, F. Maffioli, V. Maniezzo, G. Righini, and M. Trubian, "Heuristics from nature for hard combinatorial optimization problems," *International Transactions in Operational Research*, vol. 3, pp. 1-21, 1996.
- [74] L. M. Gambardella, D. Taillard, and M. Dorigo, "Ant colonies for the QAP," *Journal of the Operational Research Society*, vol. 50, pp. 167-176, 1999.
- [75] E.-G. Talbi, O. Roux, C. Fonlupt, and D. Robillard, "Parallel ant colonies for the quadratic assignment problem," *Future Generation Computer Systems*, vol. 17, pp. 441-449, 2001.

- [76] M. Solimanpur, P. Vrat, and R. Shankar, "Ant colony optimization algorithm to the inter-cell layout problem in cellular manufacturing.," *European Journal of Operational Research*, vol. 157, pp. 592-606, 2004.
- [77] M. Randall, "Near parameter free ant colony optimization," *Lecture Notes in Computer Science*, vol. 3172, pp. 374-381, 2004.
- [78] K. C. Ying and C. J. Liao, "An ant colony system for permutation flow-shop sequencing," *Computers and Operations Research*, vol. 31, pp. 791-801, 2004.
- [79] E. Taillard, "Comparison of iterative searches for the quadratic assignment problem," *Location Science*, vol. 3, pp. 87-105, 1995.
- [80] R. Susic and J. Gu, "Efficient Local Search with Conflict Minimization: A Case Study of the n-Queens Problem," *IEEE Transaction on Knowledge and Data Engineering*, vol. 6, pp. 661-668, 1994.
- [81] K. D. Crawford, "Solving the n-queens problem using genetic algorithms," *Proc. of the 1992 ACM/SIGAPP symposium on Applied computing: technological challenges of the 1990's*, pp. 1039-1047, 1992.
- [82] B. H. Gwee and M. H. Lim, "An Evolution Search Algorithm for Solving N-queen problems," *International Journal of Computer Applications in Technology*, vol. 24, 2005.
- [83] R. Susic and J. Gu, "Fast search algorithms for the N-queens problem," *IEEE Tran. On Systems, Man and Cybernetics*, vol. 21, pp. 1572-1576, 1991.

- [84] R. K. Guy, "The n Queens Problem," in *Unsolved Problems in Number Theory*, 2nd ed. New York: Springer-Verlag, pp. 133-135, 1994.
- [85] R. Sasic and J. Gu, "Fast search algorithms for the N-queens problem.," *IEEE Tran. On Systems, Man and Cybernetics*, vol. 21, pp. 1572-1576, 1991.
- [86] M. Jun and R. D. Andrea, *Path Planning for UAVs in Uncertain and Adversarial Environments, Cooperative Control: Models, Applications and Algorithms*. Kluwer, 2003.
- [87] A. Richards, J. Bellingham, M. Tillerson, and J. How, "Co-ordination and Control of Multiple UAVs," *Proc. of the AIAA Guidance, Navigation and Control Conference*, Monterey, CA, vol. 4, pp. 588-599, August, 2002.
- [88] R. W. Beard, T. W. McLain, and M. Goodrich, "Coordinated Target Assignment and Intercept for Unmanned Aerial Vehicles," *Proc. of the IEEE Intl. Conf. on Robotics and Automation*, Washington DC, pp. 2581-1586, 2002.
- [89] "Dataset: <http://www.ntu.edu.sg/home/emhlim/download/default.htm>."
- [90] W. T. Tutte, "On Hamiltonian Circuits.," *Journal of the London Mathematical Society*, vol. 21, pp. 98-101, 1946.
- [91] G. B. Dantzig and R. H. Ramser, "The Truck Dispatching Problem," *Management Science*, vol. 6, pp. 80-91, 1959.
- [92] M. L. Fisher, "Optimal Solution of Vehicle Routing Problems Using Minimum K-trees," *Operations Research*, vol. 42, pp. 626-642, 1994.

- [93] M. L. Fisher and R. Jaikumar, "A Generalized Assignment Heuristic for Vehicle Routing," *Networks*, vol. 11, pp. 109-124, 1981.
- [94] J. Kelly and J. P. Xu, "A Network Flow-Based Tabu Search Heuristic for the Vehicle Routing Problem," *Transportation Science*, vol. 30, pp. 379-393, 1996.
- [95] L. M. Gambardella, E. Taillard, and G. Agazzi, "MACS-VRPTW: A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows," in *New Ideas in Optimization*, D. Corne, M. Dorigo, and F. Glover, Eds., pp. 63-76, 1999.
- [96] B. Bullnheimer, R. F. Hartl, and C. Strauss, "Applying the Ant System to the Vehicle Routing Problem," *Proc. of the 2nd International Conference on Metaheuristics*, Sophia-Antipolis, France, July, 1997.
- [97] S. R. Thangiah, K. E. Nygard, and P. L. Juell, "Gideon: A Genetic Algorithm System for Vehicle Routing with Time Windows," *Proc. of the Seventh Conference on Artificial Intelligence Applications*, Miami, Florida, pp. 322-328, 1991.
- [98] W. Jih and J. Y. Hsu, "Dynamic Vehicle Routing Using Hybrid Genetic Algorithms," *Proc. of the 1999 IEEE International Conference on Robotics & Automation*, Detroit, Michigan, vol. 1, pp. 453-458, 1999.
- [99] J. Grefenstette, R. Gopal, B. J. Rosmaita, and D. V. Gucht, "Genetic algorithms for the traveling salesman problem," *Proc. of the First*

- International Conference on Genetic Algorithms*, J. Grefenstette, Ed.: Lawrence Erlbaum, 1985, pp. 160-168.
- [100] J. Berger and M. Barkaoui, "A Hybrid Genetic Algorithm for the Capacitated Vehicle Routing Problem," in *GECCO03, LNCS 2723*, E. Cantu-Paz, Ed. Chicago, Illinois: Springer-Verlag, 2003, pp. 646-656.
- [101] N. Christofides, A. Mingozzi, and P. Toth, "The vehicle routing problem," in *Combinatorial Optimization*, N. Christofides, A. Mingozzi, P. Toth, and C. Sandi, Eds. Chichester: Wiley, 1979, pp. 315-338.
- [102] I. H. Osman, "Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem," *Annals of Operations Research*, vol. 41, pp. 421-451, 1993.
- [103] M. Gendreau, A. Hertz, and G. Laporte, "A Tabu Search Heuristic for the Vehicle Routing Problem," *Management Science*, vol. 40, pp. 1276-1290, 1994.
- [104] L. Steinberg, "The backboard wiring problem: a placement algorithm," *SIAM Rev.*, vol. 3, pp. 37-50, 1961.
- [105] J. W. Dickey and J. W. Hopkins, "Campus building arrangement using TOPAZ," *Transportation Science*, vol. 6, pp. 59-68, 1972.
- [106] J. Krarup and P. M. Pruzan, "Computer-aided layout design," *Math. Programming Study*, vol. 9, pp. 85-94, 1978.
- [107] A. N. Elshafei, "Hospital layout as quadratic assignment problem," *Operational Res. Quarterly*, vol. 28, pp. 167-179, 1977.

- [108] M. J. Brusco and S. S. 2000, "Using quadratic assignment methods to generate initial permutations for least-squares unidimensional scaling of symmetric proximity matrices," *Journal of Classification*, vol. 17, pp. 197-223, 2000.
- [109] I. Or and K. Atik, "A quadratic assignment problem in the automated assembly of printed circuit boards," *Journal of the Operations Research Society of Turkey*, vol. 11, pp. 67-88, 2000.
- [110] E. Duman and I. Or, "The quadratic assignment problem in the context of the printed circuit board assembly process," *Computers & Operations Research*, in press.
- [111] G. Miranda, H. P. L. Luna, G. R. Mateus, and R. P. M. Ferreira, "A performance guarantee heuristic for electronic components placement problems including thermal effects," *Computers and Operations Research*, vol. 32, pp. 2937-2957, 2005.
- [112] C. S. Rabak and J. S. Sichman, "Using A-Teams to optimize automatic insertion of electronic components," *Advanced Engineering Informatics*, vol. 17, pp. 95-106, 2003.
- [113] E. M. Loiola, N. M. M. d. Abreu, P. O. Boaventura-Netto, P. Hahn, and T. Querido, "A survey for the quadratic assignment problem," *European Journal of Operational Research*, in press.
- [114] <http://www.seas.upenn.edu/qaplib/>, "QAPLIB Home Page," 2004.

- [115] N. W. Brixius and K. M. Anstreicher, "Solving quadratic assignment problems using convex quadratic programming relaxations," *Optimization Methods and Software*, vol. 16, pp. 49-68, 2001.
- [116] P. M. Hahn and J. Krarup, "A hospital facility layout problem finally solved," *Journal of Intelligent Manufacturing*, vol. 12, pp. 487-496, 2001.
- [117] P. Hahn, "Progress in solving the Nugent instances of the quadratic assignment problem," Available from: <http://wwwunix.mcs.anl.gov/metaneos/nug30/nug30.pdf>. 2000.
- [118] L. Davis, *Genetic Algorithms and Simulated Annealing*: Morgan Kaufmann Publishers, 1987.
- [119] D. E. Brown, C. L. Huntley, and A. R. Spillane, "A parallel genetic heuristic for the quadratic assignment problem," *Proc. of the 3rd International Conference on Genetic Algorithm*, Fairfax, Virginia, pp. 406-416, 1989.
- [120] T. N. Bui and B. R. Moon, "A genetic algorithm for a special class of the quadratic assignment problem," in *Quadratic Assignment and Related Problems, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 16, P. M. Pardalos and H. Wolkowicz, Eds. Rhode Island: AMS, 1994, pp. 99-116.
- [121] D. E. Tate and A. E. Smith, "A genetic approach to the quadratic assignment problem," *Computers and Operations Research*, vol. 22, pp. 73-83, 1995.

- [122] T. Mavridou and P. M. Pardalos, "Simulated annealing and genetic algorithms for the facility layout problem: A survey," *Computational Optimization and Applications*, vol. 7, pp. 111-126, 1997.
- [123] P. Merz and B. Freisleben, "A genetic local search approach to the quadratic assignment problem," *Proc. of the 7th International Conference on Genetic Algorithms*, East Lansing, Michigan, pp. 465--472, 1997.
- [124] P. Merz and B. Freisleben, "Fitness landscape analysis and mimetic algorithms for the quadratic assignment problem," *IEEE Transactions on Evolutionary Computation*, vol. 4, pp. 337-352, 2000.
- [125] J. S. Kochhar, B. T. Foster, and S. S. Heragu, "Hope: A genetic algorithm for the unequal area facility layout problem," *Computers and Operations Research*, vol. 25, pp. 583-594, 1998.
- [126] R. Tavakkoli-Moghaddain and E. Shayan, "Facilities layout design by genetic algorithms," *Computers and Industrial Engineering*, vol. 35, pp. 527-530, 1998.
- [127] D. Gong, G. Yamazaki, M. Gen, and W. Xu, "A genetic algorithm method for one-dimensional machine location problems," *International Journal of Production Economics*, vol. 60, pp. 337-342, 1999.
- [128] Z. Drezner, "The extended concentric tabu for the quadratic assignment problem," *European Journal of Operational Research*, vol. 160, pp. 416-422, 2005.

- [129] Z. Drezner and G. A. Marcoulides, "A distance-based selection of parents in genetic algorithms," in *Metaheuristics: Computer Decision-Making, Combinatorial Optimization Book Series*, M. G. C. Resende and J. P. d. Sousa, Eds.: Kluwer Academic Publishers, 2003, pp. 257-278.
- [130] M. A. El-Baz, "A genetic algorithm for facility layout problems of different manufacturing environments," *Computers and Industrial Engineering*, vol. 47, pp. 233-246, 2004.
- [131] R. L. Wang and K. Okazaki, "Solving facility layout problem using an improved genetic algorithm," *IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences*, vol. E88A, pp. 606-610, 2005.
- [132] V. Maniezzo and A. Colorni, "Algodesk: An experimental comparison of eight evolutionary heuristics applied to the quadratic assignment problem," *European Journal of Operational Research*, vol. 81, pp. 188-204, 1995.
- [133] R. E. Burkard and F. Rendl, "A thermodynamically motivated simulation procedure for combinatorial optimization problems," *European Journal of Operational Research*, vol. 17, pp. 169-174, 1984.
- [134] M. R. Wilhelm and T. L. Ward, "Solving quadratic assignment problems by simulated annealing," *IEEE Transactions on Evolutionary Computation*, vol. 19, pp. 107-119, 1987.
- [135] D. T. Connolly, "An improved annealing scheme for the QAP," *European Journal of Operational Research*, vol. 46, pp. 93-100, 1990.

- [136] P. S. Laursen, "Simulated annealing for the QAP-optimal tradeoff between simulation time and quality," *European Journal of Operational Research*, vol. 69, pp. 238-243, 1993.
- [137] N. M. M. Abreu, T. M. Querido, and P. O. Boaventura-Netto, "RedInv-SA: A simulated annealing for the quadratic assignment problem," *RAIRO Operations Research*, vol. 33, pp. 249-273, 1999.
- [138] J. Bos, "A quadratic assignment problem solved by simulated annealing," *Journal of Environmental Management*, vol. 37, pp. 127-145, 1993.
- [139] P. P. C. Yip and Y. H. Pao, "A guided evolutionary simulated annealing approach to the quadratic assignment problem," *IEEE Transactions on Systems Man and Cybernetics*, vol. 24, pp. 1383, 1994.
- [140] R. E. Burkard and E. Cela, "Heuristics for biquadratic assignment problems and their computational comparison," *European Journal of Operational Research*, vol. 83, pp. 283-300, 1995.
- [141] T. Peng, W. Huanchen, and Z. Dongme, "Simulated annealing for the quadratic assignment problem: A further study," *Computers and Industrial Engineering*, vol. 31, pp. 925-928, 1996.
- [142] P. Tian, J. Ma, and D. M. Zhang, "Application of the simulated annealing algorithm to the combinatorial optimization problem with permutation property: An investigation of generation mechanism," *European Journal of Operational Research*, vol. 118, pp. 81-94, 1999.

- [143] W. C. Chiang and C. Chiang, "Intelligent local search strategies for solving facility layout problems with the quadratic assignment problem formulation," *European Journal of Operational Research*, vol. 106, pp. 457-488, 1998.
- [144] A. Misevicius, "A modified simulated annealing algorithm for the quadratic assignment problem," *Informatica*, vol. 14, pp. 497-514, 2003.
- [145] A. Misevicius, "A new improved simulated annealing algorithm for the quadratic assignment problem," *Information Technology and Control*, vol. 17, pp. 29-38, 2000.
- [146] K. Tsuchiya, T. Nishiyama, and K. Tsujitax, "A deterministic annealing algorithm for a combinatorial optimization problem using replicator equations," *Physica D: Nonlinear Phenomena*, vol. 149, pp. 161-173, 2001.
- [147] F. Siu and R. K. C. Chang, "Effectiveness of optimal node assignments in wavelength division multiplexing networks with fixed regular virtual topologies," *Computer Networks*, vol. 38, pp. 61-74, 2002.
- [148] A. Baykasoglu, "A meta-heuristic algorithm to solve quadratic assignment formulations of cell formation problems without presetting number of cells," *Journal of Intelligent Manufacturing*, vol. 15, pp. 753-759, 2004.
- [149] J. Skorin-Kapov, "Extensions of a tabu search adaptation to the quadratic assignment problem," *Journal of Computers and Operations Research*, vol. 21, pp. 855-865, 1994.

- [150] J. Skorin-Kapov, "Tabu search applied to the quadratic assignment problem," *ORSA Journal on Computing*, vol. 2, pp. 33-45, 1990.
- [151] E. Taillard, "Robust taboo search for the quadratic assignment problem," *Parallel Computing*, vol. 17, pp. 443-455, 1991.
- [152] J. A. Bland and G. P. Dawson, "Tabu search and design optimization," *Computer Aided Design*, vol. 23, pp. 195-201, 1991.
- [153] A. Rogger, C. N. Fiechter, and D. Werra, " Basic ideas of tabu search with an application to traveling salesman and quadratic assignment.," *Ricerca Operativa*, vol. 62, pp. 5-28, 1992.
- [154] J. Chakrapani and J. Skorin-Kapov, "Massively parallel tabu search for the quadratic assignment problem," *Annals of Operations Research*, vol. 41, pp. 327-342, 1993.
- [155] A. Misevicius, "A tabu search algorithm for the quadratic assignment problem," *Computational Optimization and Applications*, vol. 30, pp. 95-111, 2005.
- [156] J. Chakrapani and J. Skorin-Kapov, "A connectionist approach to the quadratic assignment problem," *Computers & Operations Research*, vol. 19, pp. 287-295, 1992.
- [157] V. D. Cung, T. Mautor, P. Michelon, and A. Tavares, "A scatter search based approach for the quadratic assignment problem," *Proc. of the IEEE International Conference on Evolutionary Computation*, pp. 165-169, 1997.

- [158] E. Taillard and L. Gambardella, "Adaptive memories for the quadratic assignment problem," *IDSIA, Lugano, Technical Report I-87-97*, 1999.

Appendix

Table A 1. List of Abbreviation

Abbreviation	Meaning
ACO	Ant Colony Optimization
ADEP	Algorithm Development Environment for Permutation-based Problem
ANTS	Approximate Non-deterministic Tree-search
AS	Ant System
CVRP	Capacitated Vehicle Routing Problem
EA	Evolutionary Algorithm
EAML	Evolutionary Algorithm Modeling Language
EASEA	Easy Specification of Evolutionary Algorithm
FANT	Fast Ant System
FTDS	Fuel Truck Dispatch System
GA	Genetic Algorithm
GR	Genetic Representation
GRASP	Greedy Randomized Adaptive Search Procedure
HAS	Hybrid Ant Colony System
ILS	Iterated Local Search
LVRP	Left Variation-Right Property
MMAS	MAX-MIN-Ant System
PSE	Problem Solving Environment
PSO	Particle Swarm Optimization
PU	Population Update
QAP	Quadratic Assignment Problem
RWS	Roulette Wheel Selection
TSP	Travelling Salesman Problem
UAV	Unmanned Aerial Vehicle
URAV	Unmanned Reconnaissance Aerial Vehicle
VRP	Vehicle Routing Problem
VRPTW	Vehicle Routing Problem with Time Window