BUILDING OR/MS COMPUTATIONAL APPLICATIONS ON SPREADSHEET

REN XIANGYAO

(*B.Eng, Nanjing University, China*)

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF ENGINEERING

DEPARTMENT OF INDUSTRIAL & SYSTEMS ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2012

DECLARATION

I hereby declare that this thesis is my original work and it has been written by me in its entirety.

I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

Ren Xiangyao

06 Sep 2012

ACKNOWLEDGEMENTS

This dissertation will not have been possible without the generous help, encouragement and support of a number of people to whom I owe my great thanks and appreciation over the past three years. It was my pleasure to study and work with them.

First and foremost, I owe my particular gratitude to my supervisor, Professor Teo Kwong Meng, for his invaluable guidance and kindly support throughout the entire period. With his help, I have been able to correct from wrong and learn from failure. His insightful ideas, rigorous thoughts, and great enthusiasm inspired me and made this research a precious experience in my life, and I believe such experience will continually benefit me for the whole life.

Besides, I would like to thank the National University of Singapore for offering me the Research Scholarship. I owe my great thanks to Prof Tang Loon Ching and Prof Lee Loo Hay, who has offered great support for me to complete this research work. Without their generosity and patience, this research work will not have been possible. I am very grateful to my colleagues in ISE Department for their kind help. Especially, I would like to thank Lai Chun, for her every patience and help for my questions. I owe my great thanks to all the people who have helped me in one way or the other.

I feel deeply indebted to my family for their endless love, support and encouragement. My wholehearted thankfulness and gratefulness goes to my girlfriend Jing Hua Yi, who has provided me the best and priceless love, encouragement, and support to facilitate the completion of this thesis. I will remember my deepest appreciation to my great love in my mind and this thesis is dedicated to you.

Ren Xiangyao

January, 2012

TABLE OF CONTENTS

SUMMARY

With the great usage of spreadsheet in business and scientific world nowadays, building computational applications on spreadsheet has become essential for people to conduct data analysis, algorithm computation, and solving problems. However, when different options of spreadsheet software and methods are available to build spreadsheet applications, the performance difference of these options and how to make the selection is rarely addressed. The purpose of this research is to investigate the performance and implementation effort of different options, provide guidelines for people to select between different options, and provide people with an easier start to build applications on spreadsheet.

We define the Internal, External and Hybrid programming methods of building computational applications on two of the most popular spreadsheet software: Microsoft Excel and OpenOffice.org Calc. A comprehensive performance comparison of different methods on these two spreadsheets is conducted, from which the insights of the performance differences of different options under different scenarios and the ease of implementation of different options are revealed.

Based on the comparison, we construct a framework of building computational spreadsheet applications that provides guidelines of selecting between different options under different criteria. Comprehensive implementation examples in areas of operations research and management science, such as Sort, Shortest Path, TSP and VRP spreadsheet applications are built with structural routines and library codes. It is able to provide people with an easier start to build spreadsheet applications.

With the framework of building computational applications on spreadsheet, people can apply the most effective approaches based on their requirements and build spreadsheet applications with much more convenience and efficiency.

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

Algo	Algorithm computing
API	Application Programming Interface
BIFF	Binary Interchange File Format
Calc	OpenOffice.org Calc
COM	Component Object Model
DLL	Dynamic Link Library
Excel	Microsoft Excel
IDE	Integrated Development Environment
MS	Management Science
ODF	Open Document Format
OLE	Object Linking and Embedding
OOO Basic	OpenOffice.org Basic
OR	Operations Research
Read	Read data from spreadsheet
RTD	Real Time Data
SDK	Software Development Kit
TSP	Travelling Salesman Problem
VBA	Visual Basic for Applications
VBE	Visual Basic Editor
VC++	Microsoft Visual C++
VRP	Vehicle Routing Problem
VRPTW	Vehicle Routing Problem with Time Windows
Write	Write result to spreadsheet
XML	Extensible Markup Language

Chapter 1

Introduction

1.1 MOTIVATION

1.1.1 Background

The spreadsheet discussed in this paper refers to the computer software that simulates paper accounting worksheets. The main concepts are those of a grid of cells, called sheet, storing either raw data values or formulas in the cells. An array of cells is analogous to an array of variables in a conventional computer program.

The first electronic spreadsheet, VisiCalc, became an instant success when it was introduced in 1978 (Power 2004). Since it was introduced, spreadsheet embraces an explosive growth. It has become a ubiquitous tool that is used in almost all business work and scientific fields. Hesse and Scerno (2009), based on their almost 20 years of experience of using spreadsheets, share their perspectives that spreadsheet has tremendously changed the world in various ways such as in computer usage, training and education, data analysis and presentation. Spreadsheet is widely used today due to its great advantages listed below:

- Automatic calculation: with the cell references in spreadsheet, the data value can be recalculated automatically. Users only need to change the entry with a new value, and the cells will be updated. This highly improves the efficiency.

- Diverse formatting and charting: spreadsheets allow users to format the data in various appearances and present them with various types of graphs. With this property, data can be better presented and interpreted.

- Data integrity: spreadsheet is able to check the consistency of data automatically. It will reject or correct wrong entries to enforce the data integrity.

1.1.2 Importance of the Study of Building Applications on Spreadsheet

Spreadsheet is ubiquitous due to its general availability, accessibility and ease of use (Rosen and Adams 1987). Based on data stored in spreadsheet, people can build applications on it to conduct various kinds of data analysis and decision support analysis (Ragsdale 2011), compute results and solve problems directly. Building computational applications on spreadsheet becomes essential both in business industry and scientific fields nowadays. For example, in Engineering and industries, researchers can build spreadsheet applications to carry out specific computations and solve problems (Fields 1986, Whitehouse and Hodak 1986, Jr 1987, Kokol 1989, Zimmerman and Gibson 1989, Bloch 1995, Dianond and Hanratty 1997, Billo 2011). In Business, people can build spreadsheet applications to carry out data analysis and simulation (Earnest 1987, Raffensperger 2003). These spreadsheet applications with computational usage further extend the capability of spreadsheet and help people obtain solutions conveniently and analyze data efficiently.

One of the well-known examples is the Excel Solvers built by Frontline Corporation. It can solve various kinds of optimization problems such as conventional optimization, simulation optimization and stochastic optimization problems. Rosen and Adams (1987) and Chehab et al. (2004) review the spreadsheet applications in Chemical Engineering and Electrical Engineering respectively. They conclude that in various computational instances, building applications on spreadsheet is an important and attractive alternative compared with other means of computational applications. Filby (1998) introduces abundant research examples in Science and Engineering on building applications and models on spreadsheet. Oke (2004) reviews the applications built on spreadsheet in Engineering Education and points out the importance of applications on spreadsheet to the need of high quality, learning-centered education.

1.1.3 Methods used to Build Applications on Spreadsheet

Among various kinds of spreadsheet software, Microsoft Excel is the most successful spreadsheet software which dominates the commercial market and owns over one billion users

worldwide (Stan J. Liebowitz 2001, Ionut Arghire 2012). Almost all standard entrepreneurship textbooks propose Excel spreadsheets to create a financial plan as part of a business plan (Gansel 2008). Meanwhile, OpenOffice.org Calc is a free and open-source spreadsheet software modeled after Excel. These two kinds of spreadsheet are the most popular spreadsheet software people used nowadays.

There are different kinds of methods available to build computational applications on spreadsheet, which can be classified into two parts: Built-in functions and Programming methods.

Built-in functions are the functions integrated in spreadsheet software, such as *Sum()* function. They can be used to carry out simple calculations and solve simple problems iteratively, such as Sort, Solver, etc. The data in spreadsheet Cells are referenced and passed to Built-in functions, and then the computation results are displayed in Cells where the Built-in functions are used. When the problem is relatively simple and problem size is small, Built-in function is a very convenient method to build computational spreadsheet applications. However, when the problem size and difficulty increases, Built-in functions will become extremely difficult to implement or impossible to use. For example, when the number of decision variables exceeds 200, the Standard Solver in Excel spreadsheet is not applicable.

Thus, for larger size and more complicated problem, programming methods are applied to build spreadsheet applications to read input data from spreadsheet, compute output results, and then write output results back to spreadsheet (Hazel n.d., Walkenbach 2004). Based on where the data transferring and computing process are located, we categorized the programming methods into three groups, as defined below:

1. Internal programming methods: Internal programming methods are programming methods which are integrated with IDEs inside spreadsheet software, such as VBA (Visual Basic for Applications) in Excel, and OOO Basic (OpenOffice.org Basic) in Calc. As shown in Figure 1.1, the arrow shows the data flow. Data can be read from spreadsheet into Internal

programming methods, then computing process is conducted, and the results obtained are written back to spreadsheet. It allows users to define the algorithm and calculation steps themselves. At the same time, the Built-in functions can also be called in macros written by Internal programming methods. Because of its powerfulness and convenience, this method has become the most popular method to build computational spreadsheet applications.

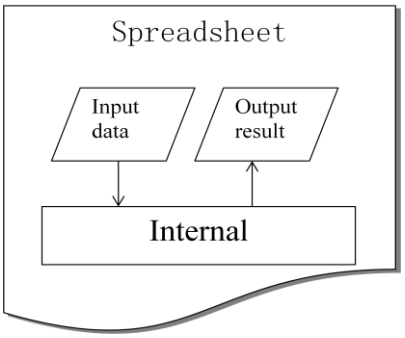

Figure 1.1 Internal Programming methods to build spreadsheet applications

2. External programming methods: These methods, such as C++ and Java, can be used to access spreadsheet and build applications on it. As shown in the Figure 1.2, spreadsheet Input data can be read into External programming methods, where computing process is carried out, and then results can be written back to spreadsheet.

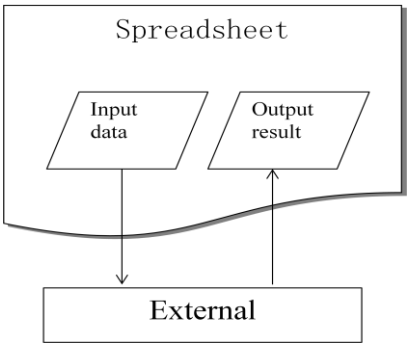

Figure 1.2 External Programming methods to build spreadsheet applications

3. Hybrid programming methods: Hybrid programming method is the combination of Internal and External programming methods to build computational applications on spreadsheet, such as VBA call C++/FORTRAN DLL method combining VBA and C++/FORTRAN. As shown

in the Figure 1.3, Input data on spreadsheet are read by Internal methods (VBA) and passed to External methods (C++/FORTRAN DLL) to carry out computing process, and then results are passed from External methods (C++/FORTRAN DLL) to Internal methods (VBA) and written back to the spreadsheet. Rosen and Partin (2000) introduce the VBA call FORTRAN DLL method to convert the existing standalone FORTRAN programs to applications on Excel.

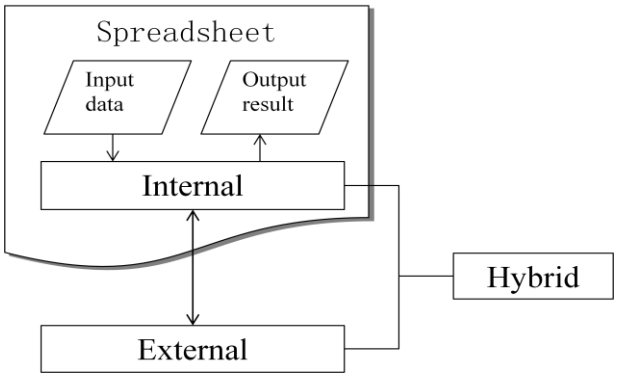

Figure 1.3 Hybrid programming methods to build spreadsheet applications

In this thesis, we focus on Excel and Calc as they are the most popular spreadsheets used nowadays. Based on these two spreadsheet software, we will focus on using Internal, External, and Hybrid programming methods to build computational spreadsheet applications. Specifically, for Internal programming method, we select VBA on Excel, OOO Basic on Calc as they are integrated in the respective spreadsheet software. For External programming method, we select VC++ (Visual studio C++), Java on Excel and Java on Calc as they are the most popular and typical methods in use to build spreadsheet applications nowadays. For Hybrid programming method, we select VBA calling C++ DLL on Excel which is a natural combination of Internal and External methods, and to our best knowledge, Hybrid programming method on Calc is not feasible and hence will not be discussed in this study. The reason will be discussed in Chapter 3 section 3.1.

1.1.4 Motivation of this Research

Although there are numerous studies on how to build computational applications on spreadsheets for solving problems, however, a fundamental issue -- the performance difference,

which refers to the speed of applications on spreadsheet, among various building methods are rarely addressed. The previous research works tell us the feasibility of using different options to build applications on spreadsheet. However, knowing their feasibility is not equivalent to knowing the strengths and weaknesses of the methods and the spreadsheet software. Thus, an inappropriate method, which is inefficient for certain scenarios, can be chosen by researchers and practitioners. Such inefficient choice will result in a huge waste of critical resources (e.g., human skills, man-hours, Information technology (IT)), which leads to producing much more costs in industries.

Therefore, the knowledge of performance differences among various options is critical for making the right decisions in different scenarios. This is the essential motivation of this study. We intend to comprehensively investigate the performance, in terms of speed, of different implementation methods, as well as their ease of implementation, and help people to select the most efficient method among different options at the earliest stage of building spreadsheet applications. In short, the goal is to make it easier to deal with problems of building computational applications on spreadsheet.

1.2 RESEARCH DESIGN

1.2.1 Research Objective

In section 1.1, we discuss the importance of building computational applications on spreadsheet. We observe that making the right choice among different options of building spreadsheet applications is critical. Therefore, we will study on building spreadsheet applications at both strategical and tactical levels. Based on the discussions above, we formulate our research objective as follows:

To investigate the performance differences and ease of implementation of different options to build computational spreadsheet applications, provide guidelines of selecting the most efficient option among them under different scenarios, and provide people with a much easier and quicker start of building spreadsheet applications.

We discuss the important components of our research objective below:

1. *Performance difference*: We aim to show the performance difference, specifically, the speed difference of spreadsheet applications built with different methods. Besides, we aim to tell the ease of implementation of each option.

2. *Guidelines*: Based on the performance differences, we aim to provide the guidelines of how to select the fastest method with the least implementation effort among various options. Thus, people can make the most efficient selections under different requirements.

3. *Easier and quicker start*: After people selecting a specific option to build spreadsheet applications, we strive to reduce the people's amount of work of implementing this specific option to build spreadsheet applications by providing structural routines and library codes, and thus the efficiency and convenience of the implementation process can be greatly improved.

1.2.2 Research Questions and Approaches

To achieve our objective, we propose a number of research questions that have to be answered. The logical sequence of research activities are also reflected in these questions. For each question, we discuss the approach we are going to use to arrive the answer.

Q1: What are the performance differences of different methods on spreadsheet? (Chapter 3)

To answer this question, we comprehensively compared the performance of different methods on Excel and Calc spreadsheets using implementation tests with growing problem size and algorithm complexity. For implementation tests, we select Sort, Shortest Path and TSP to be our test problems due to their growing difficulty and complexity. For each test problem, we select small, medium and large levels of problem size to implement. We compare the running time of implementations in the aspects of total, data transferring and algorithm computing to reveal the performance difference of different methods on spreadsheet in these aspects.

Q2: What is the implementation effort required to build spreadsheet applications using different methods? (Chapter 3)

However, even when Q1 is answered, people will still want to know the development effort of different methods to build spreadsheet applications. This is because if the performance between different methods makes no difference in certain situations, people will certainly like to use the method with the least implementation effort. Assuming people are unsophisticated developers or inexperience programmers, we present the implementation effort of different methods in terms of the code structures and the amount of codes needed to build up the applications on spreadsheet. Through the construction of comprehensive implementation tests using different methods on spreadsheet, we are able to tell the ease of implementation of different methods to build spreadsheet applications.

Q3: To what extent can we build computational spreadsheet applications to solve very complicated problems? (Chapter 4)

Even if people are aware of the strengths and weaknesses of different options, it remains to investigate the capability of computational spreadsheet applications to solve very complicated problems. The lack of such upper bound information on capability can result in the underestimation of the capability of spreadsheet applications. To answer this question, we construct a spreadsheet VRPTW application using the best method from the answers of Q1. We apply the tabu-search heuristics (Lau et al. 2003) to solve the VRPTW problem, the VRP problem with Time Windows. We use this spreadsheet VRPTW application to solve all the 56 Solomon test cases, which are well-established benchmark test cases for VRPTW problem (Solomon, 1987), and compare the performance with a C++ standalone application reading and writing data on text files. To this end, by solving a very complicated problem with sophisticated heuristics using a spreadsheet VRPTW application and comparing its performance with standalone computational applications, we can obtain the insights of the capability of computational applications based on spreadsheet.

Q4: How to select between different options of building computational spreadsheet applications? (Chapter 5)

Based on the performance differences and ease of implementation of different methods on spreadsheet from Chapter 3, we construct a framework of building applications on spreadsheet that provides guidelines of selecting between different options under different scenarios. We study and identify the criteria to select between different spreadsheet software and different methods following the sequence of building spreadsheet applications, and under each scenario, we provide the most efficient option with the least implementation effort. With such a framework, people can select the most efficient way among various options to build computational spreadsheet applications based on their requirements.

Q5: How to make it easiest to build computational applications on spreadsheet? (Chapter 5)

With the framework providing guidelines of selecting between different options, people can select the most efficient methods to build applications on spreadsheet based on their requirements. We construct the structural routines of different methods and the library codes of comprehensive implementation examples to further provide people with an easier start to build spreadsheet applications with specific method. In this way, we strive to reduce the amount of work to build spreadsheet applications to the largest extent and improve the convenience and efficiency involved.

1.3 RESULTS AND CONTRIBUTIONS

1.3.1 Principal Results

Our principal result is the framework of building applications on spreadsheet. It provides guidelines of selecting the most effective method to build computational spreadsheet applications under different scenarios, as shown in Figure 1.4 below.

Through our principal result, the framework of building spreadsheet applications, the research questions can be answered and our research objective can be achieved.

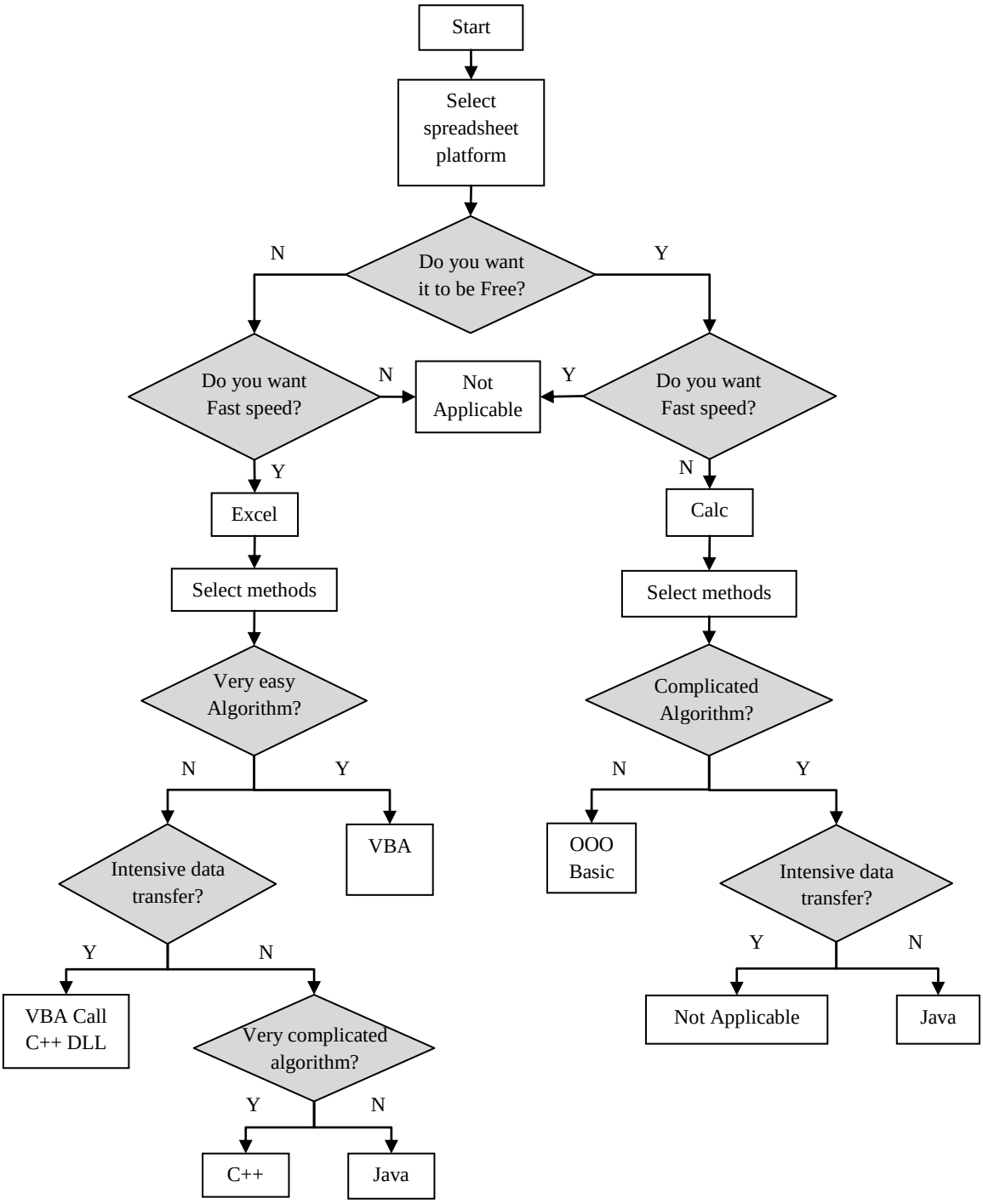

Figure 1.4 Framework of building applications on spreadsheet

1.3.2 Our Contributions

Our contributions to the research study of building computational applications on spreadsheet in this thesis are summarized below:

1. A framework providing guidelines of selecting between different methods to build computational applications on spreadsheet; with this framework, people are able to apply the most efficient approach to build spreadsheet applications based on their requirements, which supports the decision making and improves the efficiency.

2. The ease of implementation analysis of different methods to build spreadsheet applications; for unsophisticated developers or inexperienced programmers, it is very important to have the information of implementation effort of different options so that they can better plan and schedule their resources to build applications on spreadsheet. With the ease of implementation analysis, people can choose the one with the least implementation effort from feasible options.

3. An Excel VRPTW application with good performance, which has not been found in the literature; with the insights of capability of spreadsheet applications to solve the VRPTW problem, people are able to tell the feasibility of building computational spreadsheet applications to solve very complicated problems.

4. A Sync process to transfer data arrays with dynamic length which overcomes the inherent limitation of VBA call C++ DLL method to build spreadsheet applications; with such a process, the data arrays with dynamic length can be transferred between VBA and C++ DLL by being transformed into data arrays with static length.

5. Structures, routines and library codes of different methods to build spreadsheet applications; it provides people with a much easier start to build applications on spreadsheet. With the structural routines and library codes, people are able to conveniently and easily follow the routines and sample codes to build applications on spreadsheet with the specific option they have selected, which saves the cost and improves the efficiency greatly.

1.4 THESIS ORGANIZATION AND STRUCTURE

In this research framework, this thesis is organized and structured as follows:

Chapter 2 contains the literature review on building computational applications on spreadsheet. In this chapter, recent literatures on building computational spreadsheet applications will be reviewed and the limitations of current studies will be discussed to form the basis of motivation of this research.

In Chapter 3, we conduct a comprehensive comparison of the performance of different methods on spreadsheet and their ease of implementation analysis. The running time of implementation tests built on spreadsheet with increasing algorithm complexity and problem size are compared. Next, the amount of implementation effort in terms of the code structures and the codes needed for different methods on spreadsheet are compared. Through these comparisons, the strengths and weaknesses of each method under different criteria and the ease of implementation of each method will be concluded.

In Chapter 4, we build an Excel VRPTW application using VBA call C++ DLL (Hybrid) method to show that a spreadsheet application solving a complicated problem with sophisticated heuristics can be successfully built using the VBA call C++ DLL method. Meanwhile, the Excel VRPTW application performance is compared with a C++ standalone application under all 56 Solomon test cases. The insights of the capability of using VBA call C++ DLL method to build computational spreadsheet applications will be revealed.

Chapter 5 will construct the framework of building computational applications on spreadsheet. Through the comparative study on different spreadsheet software and different methods of building applications on spreadsheet, we will provide guidelines of selecting between different options under different scenarios. Furthermore, we will provide structured routines and library codes of different options to provide people with an easier start.

In Chapter 6, we give concluding remarks and discuss the limitations and possible further extensions of this research.

Chapter 2

Literature Review

2.1 INTRODUCTION

As discussed in Chapter 1, there are plenty of studies about how to build applications on spreadsheet in order to solve different problems. Therefore in this chapter, we discuss some published research and development of spreadsheet applications that are related to our research topic. Firstly, we describe some general literatures on computational spreadsheet applications built for solving different problems, especially in science and engineering areas (section 2.1). Next, we discussed different methods used to build computational spreadsheet applications (section 2.2). In section 2.2.1, the use of built-in functions and solvers is addressed. The spreadsheet applications using three different types of programming methods are reviewed separately in sections 2.2.2, 2.2.3 and 2.2.4. Finally, we summarize our findings from the literature and discuss how these interesting findings motivate this research (section 2.3).

2.1 DIFFERENT SPREADSHEET SOFTWARE USED FOR BUILDING APPLICATIONS

The spreadsheets are used widely because of their obvious advantages in terms of efficiency, various ways of formatting, data integrity, and automatic and accurate charting generation. Since the first electronic spreadsheet VisiCalc was developed, many different kinds of spreadsheet software have been invented and the spreadsheet software market is maturing. These spreadsheets can be classified into two categories: online spreadsheets and desktop spreadsheets (Obrenovic and Gasevic 2008). Google spreadsheets are one of the most popular online spreadsheets that can be accessed from the Google Docs. The desktop ones include Microsoft Excel and some open-source software like OpenOffice.org Calc. Microsoft Excel is the most successful commercial spreadsheet software. Starting from 1995 to the present, Microsoft Excel has dominated the commercial spreadsheet market (Stan J. Liebowitz 2001).

During the last two decades, numerous research studies have been conducted to build applications and models on spreadsheet for Business usages, Engineering calculations and Engineering educations (Rosen and Adams 1987, Chehab et al. 2004 ,Oke 2004). Among these studies, when the spreadsheet software is referred, Microsoft Excel is the most popular spreadsheet software used to build applications in Science and Engineering. Moreover, Excel spreadsheets are proposed to be used in most of the standard entrepreneurship textbooks to create financial plans in the business world (Gansel 2008). However, OpenOffice.org Calc, which is an open-source and free spreadsheet software developed after Microsoft Excel, has also become a very popular and important spreadsheet software nowadays. It is able to run on various kinds of Operating Systems including Windows, Mac OS and Linux.

Thus, having observed the popularity of Excel and Calc, we will choose them as the spreadsheet platforms of our research. Since Calc is a free and popular spreadsheet software, and there is little research on spreadsheet applications on Calc, it will be worthwhile to extend applications from Excel spreadsheet to Calc. This gap of building applications on spreadsheets other than Microsoft Excel in Science and Engineering research areas leads to one of our motivations for this research framework.

2.2 DIFFERENT METHODS OF BUILDING APPLICATIONS ON SPREADSHEET

2.2.1 Built-in Functions and Solvers

Many researches on spreadsheet models and applications apply cell reference functions, and built-in formulas inside the spreadsheet software to carry out engineering calculations iteratively to achieve the computation results. Archer (1989) applies the digraphs to represent the logical ordering of the cell reference calculations. For problems with iterative solutions, a pseudo cell relationship diagram (CRD) is generated to present the cell references and data flows in this kind of computational applications on spreadsheet. Anthony and Wilson (1990) build a simple manpower model on SuperCalc spreadsheet system with Cell references to

solve the problem step by step. It is shown that using Built-in functions to build applications on spreadsheet is useful for developing models and insights rapidly and for producing unsophisticated results. However, once the model starts to grow, it will outgrow the spreadsheet approach. Filby (1998) introduces abundant examples of applications in Science and Engineering using Built-in functions to build computational applications on spreadsheet.

According to Rosen and Adams (1987), the applications built with Built-in functions on spreadsheet have some advantages. For example, the user can define tabular format, and the arithmetic operations are hidden from input and output results. Hence they are user-friendly and easy to use. The calculations are carried out step by step and non-procedural, and hence will be very intuitive and easy to understand. However, this kind of application built with Built-in functions on spreadsheet has a number of limitations. Firstly, the spreadsheet application is calculated through the entire process for each update. Secondly, no return capabilities are available to carry out a subroutine or recursive type of calculation. Thirdly, when the problem becomes very difficult and data size becomes very large, the applications on spreadsheet using built-in functions are very tedious to build.

Frontline Solvers is the developer of the Standard Solver in Excel spreadsheet. Standard Excel Solver is able to solve Linear and Non-linear problems with a maximum size of 200 variables. It is applicable across different areas and able to solve various problems (Frontline), such as portfolio optimization in Finance and Investment, job scheduling in Manufacturing, routing and loading in Distribution and Networks, etc. Lynne and John (2004) apply the Excel solver on real sample design problems with complex features, and also discussed about other solver tools which are widely available nowadays in spreadsheet for solving sample design problems. However, to build applications on spreadsheet with Built-in Solvers, the problem size will be limited, and the algorithm cannot be user-defined.

2.2.2 Internal Programming Methods

As discussed in section 2.2.1, both built-in functions and solvers have some limitations. Since spreadsheet is a powerful programming language and is viewed as the “fourth-generation

programming language" (Thomas, A. Grossman 2010), when the built-in functions and solvers are not able to satisfy the needs of researchers and engineers, they start to apply Internal programming languages integrated inside the spreadsheet in order to get a more powerful option.

For problems with more complicated algorithm and larger problem size, many literatures have proposed building computational applications on spreadsheet using Internal programming languages. With the advent of VBA, which is a programming method integrated in Excel spreadsheet, researchers can write macros using VBA to build computational spreadsheet applications. LeBlanc and Galbreth (2007) describe an efficient way which uses VBA in Excel to solve Large-Scale linear optimization problems (LPs). The model built on Excel with VBA has overcome limitations for large-scale problems and increased model usability. Au et al. (2010) develop a prototype VBA package implementing advanced Monte-Carlo simulation which is able to perform efficient uncertainty propagations. David and Ragsdale (2003) improve the Excel solver with VBA to solve stochastic multi-criteria linear problems. Numerous applications built with VBA on spreadsheet in Science and Engineering can also be found in Filby (1998).

2.2.3 External Programming Methods

External programming languages, such as C/C++, Java, Visual Basic, C#, can also be used to build computational applications on spreadsheet. External programming methods are different from Internal programming methods, they are not integrated in spreadsheet and hence they are not able to build applications on spreadsheet directly. Literatures are available on building up the interface between spreadsheet and External programming methods. For instances, Hazel introduces the method of accessing Excel spreadsheet from within C++ using Microsoft Component Object Model (COM). Sakalli and Birgoren (2009) developed spreadsheet-based decision support tools that link Excel with LINGO modeling language and optimizer. LINGO is a comprehensive tool designed to build and solve various kinds of optimization models, such as Linear, Non-linear optimization, etc. LINGO modeling language is integrated in the

package for expressing optimization models. It is able to import data from spreadsheets and export solutions back out to spreadsheets through the OLE (Object Linking and Embedding) links. LibXL is a C++ library that can read and write Excel files. J-Integra is a Java interoperability component that bridges Java and Microsoft Excel. JXL is a free open-source java API enabling developers to read and write Excel spreadsheets.

2.2.4 Hybrid Programming Methods

Hybrid programming methods are the hybrid of Internal programming methods and External programming methods or packages. Rosen and Partin (2000) first propose a method to utilize the existing standalone FORTRAN programs in the spreadsheet environment with VBA. For example, in this research work, VBA is able to utilize the FORTRAN code by declaring the function in FORTRAN as a Dynamic Link Library (DLL), which makes the process simple and convenient, and at the same time also proves the strong capability of spreadsheet. Rosen (2001) also describes a method of VBA call C++ DLL on spreadsheet to carry out calculations with simple examples. Additionally, Punuru and Knopf (2008) introduce the concept of linking VBA with C/C++ code. They systematically present on how to facilitate the transfer of data, such as single variables, vectors and matrices, between VBA and C++ DLLs with various illustrative examples. Hazel also introduces VBA call C++ DLL method to build computational applications on Excel spreadsheet. Moreover, Frontline's Risk and Premium Solver software is developed using External programming languages and can be deployed in VBA as XLL add-ins. This Solver software is able to solve various kinds of optimization problems of large size with algorithms. However, it lacks the ability to allow users to solve problems with self-defined heuristics.

2.3 SUMMARY

In this Chapter, different spreadsheet software and methods used to build computational applications on spreadsheet in Business, Science and Engineering are substantially reviewed. Nowadays, numerous applications have been built on spreadsheet with various kinds of methods. This includes for instance, applying Built-in functions in spreadsheet to carry out

calculations and solve problems iteratively, and applying Internal, External and Hybrid programming methods, such as VBA, C++, VBA call C++ DLL, to build more capable and efficient applications on spreadsheet for more complicated and larger size problems. However, there are several questions and topics that need more investigation, thus motivate the studies involved in this dissertation.

Firstly, the current studies on building applications on spreadsheet are mostly based on Excel spreadsheet software. OpenOffice.org Calc, which is an open-source and free spreadsheet software popularly used around the world nowadays, will also be of great value and interest to be investigated to build computational applications on it.

Secondly, with different methods to build computational spreadsheet applications, an important research question is to find out the performance difference among these methods. This knowledge is critical for people to make decisions when several options are available to be selected. Besides, the ease of implementation of these methods is also very important to build spreadsheet applications. However, these topics are rarely addressed in literatures on spreadsheet applications. Chapter 3 is motivated to answer this research question.

Thirdly, another important question is how capable is the spreadsheet application to solve very complicated problems with sophisticated heuristics. Without such information, people will not be able to easily tell the feasibility of building computational spreadsheet applications for complicated problems with sophisticated heuristics. Chapter 4 is motivated by attempting to fill this gap.

Lastly, facing different choices and combinations of spreadsheet software and different methods, how to select among these options to build applications on spreadsheet under different scenarios is rarely addressed. If people could select the most suitable method to build computational spreadsheet applications in different situations, the efficiency will be greatly improved. This is the research objective of Chapter 5.

Chapter 3

Performance Comparison of Different Methods on Spreadsheet

3.1 INTRODUCTION

As discussed in Chapter 2, there are numerous studies on how to build computational applications on spreadsheet. However, the performance differences, in terms of the speed of computational spreadsheet applications, among various building methods, and their ease of implementation are rarely addressed. The strengths and weaknesses of different options are the most critical information for selecting within different options to build computational applications on spreadsheet in different scenarios. Without such knowledge, the most efficient method may not be selected among various options. Moreover, with the knowledge of implementation effort of different options, people can better plan and schedule resources to fulfill the developing task.

Therefore, in this chapter, we will focus on two of the most popular spreadsheet software, Excel and Calc. Based on these two spreadsheet platforms, Internal, External and Hybrid programming methods, including VBA, VC++, Java, VBA call C++ DLL and OOO Basic are used to build implementation tests on spreadsheet, to compare their performance and analyze their ease of implementation. The reason for choosing these options is because of their importance and popularity in building computational spreadsheet applications. The combinations of software and methods are summarized in Table 3.1 shown below.

We organize Chapter 3 as follows: Firstly, to investigate the performance difference of different methods on spreadsheet, we believe the best way is through comprehensive comparison tests. In section 3.2, we describe different testing problems used in this chapter. Based on this, we build implementation tests using different methods with increasing

algorithm complexity and problem size to compare their performance (sections 3.3, 3.4). For example, we select merge-sort to solve Sort problem, Dijkstra for Shortest Path problem and 2opt and 3-opt heuristics to solve TSP problem. The performance measure refers to the speed of spreadsheet application. Hence, the running time of each implementation test is compared in three aspects: Total time, Data transferring time, and Algorithm computing time.

Table 3.1 Options of spreadsheet software and methods in this research framework

	Microsoft Excel	OpenOffice.org Calc
Internal Programming Method	VBA	OOO Basic
External Programming Method	VC++, Java	Java
Hybrid Programming Method	VBA call C++ DLL	*

*: *Hybrid methods on Calc are not discussed in this research study as data is not stored consecutively in OOO Basic, and the data array is not able to be passed by address in the same ways as Hybrid method on Excel such as VBA call DLL to build spreadsheet applications. Hence, the Hybrid method on Calc is considered not feasible in this research framework.*

Secondly, to investigate the ease of implementation of different methods on spreadsheet, we assume that the implementation effort required is in terms of the code structures and amount of codes required to write for unsophisticated developers or inexperienced programmers. Based on this assumption, we will illustrate the code structures and the critical codes of different methods to build spreadsheet applications. In section 3.5, we analyze the ease of implementation of different methods on the two spreadsheet platforms. We conclude this Chapter in section 3.6.

3.2 TESTING PROBLEM DESCRIPTION

In order to investigate and compare the performance of different methods on spreadsheet comprehensively, it is important to construct the implementation tests in two dimensions: algorithm complexity and problem size.

For algorithm complexity, the Merge-sort algorithm for Sort problem, the Dijkstra algorithm for Shortest Path problem and the 2-opt+3-opt heuristic algorithm for TSP (Travelling Salesman) problem are selected to build implementation tests due to their growing difficulty and complexity. Hence with increasing algorithm complexity, the algorithm computing time

performance of different methods on spreadsheet can be compared comprehensively. For problem size, each specific problem is implemented with growing data size to compare the data transferring speed performance of different methods on spreadsheet. Therefore, the implementation tests are able to reveal the performance of different methods comprehensively from problem with easy algorithm and small size to problem with complicated algorithm and large size.

The basic information of implementation tests can be summarized in Table 3.2 shown below.

Table 3.2 Description of implementation tests on spreadsheet

				Problem Size		
Test Problem	Algorithm	Algorithm Complexity		Small	Medium	Large
Sort	Merge-sort	Easy	$O(n \log n)$	1000	10000	50000
Shortest Path	Dijkstra	Medium	$O(n^2)$	500	1000	5000
TSP	2-opt + 3-opt	Complicated	$O(n^3)$	50	100	150

Implementation tests to compare the performance of different methods on spreadsheet are illustrated in detail below.

For the Sort problem, the Merge-sort algorithm is used to compute the result. Merge-sort is a very efficient sorting algorithm and the time complexity is $O(n \log n)$. For each run, the Input data is randomly generated and stored as a single column, and the Output result will be computed and sorted also in a single column in the same spreadsheet. This can be shown in Table 3.3 shown below.

Table 3.3 Input and Output formats in Sort implementation test

Input	Output
Data[1]	Result[1]
…	…
Data[n]	Result[n]

For the Shortest Path problem, the problem used in this research is the single-source shortest path problem, and the classic Dijkstra algorithm is used to obtain the solution. The Dijkstra

algorithm is the most popular algorithm used to solve the single-source shortest path problem, and its time complexity is $O(n^2)$. In this study, the Input data is the Graph adjacency matrix which indicates the Path cost if there is an Edge between Vertex i and j. The output is the shortest path to each location from the single source Vertex *0* and the total cost of each shortest path. For each size of the implementation test, the adjacency matrix is randomly generated and stored in three columns in the spreadsheet. The Output result is computed and stored in the same spreadsheet. This can be shown in the table below.

Table 3.4 Input and Output formats in Shortest Path implementation test

Input			Output		
Vertex *0*	Vertex *i*	Cost[0][i]	Vertex *0*	Total Cost[0]	Path (0 to 0)
…	…	…	…	…	…
Vertex *j*	Vertex *n*	Cost[j][n]	Vertex *n*	Total Cost[n]	Path (0 to n)

For the TSP problem, it is a NP-hard problem which is not able to obtain optimal solution in polynomial time. In this study, a combination of 2-opt and 3-opt heuristics to obtain an approximation of the optimal solution is used. 2-opt and 3-opt will iteratively improve the solution in each iteration, its time complexity is $O(n^3)$ and the combination of 2-opt and 3-opt will increase the probability of finding better approximation to the optimal solution. The Input data is the randomly generated coordinates of all points from 0 to n stored in two columns in the spreadsheet. The Output result is the approximate optimal solution of the shortest Hamiltonian cycle route and its total distance. It is computed and stored in the same spreadsheet. This can be shown in the table below.

Table 3.5 Input and Output formats in TSP implementation test

Input		Output	
X[0]	Y[0]	Total Distance	Route[0]
…	…		…
X[n]	Y[n]		Route[n]

All implementation tests built on spreadsheet will follow the 3 general steps shown below:

(1) Load file and Read data from spreadsheet;

(2) Solve the problem using specific algorithm;

(3) Write result to spreadsheet and save file;

In the following research framework, these 3 steps will be denoted as Read, Algo, and Write for short. The running time of each part will be recorded separately and compared.

All the tests were carried out under the same condition on the same computer device (Intel Core2 Duo CPU, T9600, 2.8GHz, RAM 4.00GB, 32-bit Windows Operating System). Each specific problem is repeated with 100 Runs to obtain the average running time results.

3.3 PERFORMANCE COMPARISON OF DIFFERENT METHODS ON EXCEL

Through the comprehensive implementation tests built on Excel spreadsheet, we are able to compare the performance of different methods on Excel spreadsheet in following sub-sections.

To restate the different methods on Excel spreadsheet, we select 4 different methods and compare their performances on Excel spreadsheet. They are VBA, VC++, Java, and VBA call C++ DLL. Among them, VBA is the Internal programming method, VC++ and Java are the External programming methods, and VBA call C++ DLL is the Hybrid programming method. To investigate and compare their performances, we apply these programming methods to build implementation tests on Excel spreadsheet to read and write on spreadsheet and carry out algorithm computations.

In sub-sections 3.3.1 to 3.3.4, we will present the performance results of the 4 methods on Excel. In sub-section 3.3.5, we will compare their performances. In sub-section 3.3.6, we will give a summary.

3.3.1 Performance of VBA on Excel

The running time results of implementation tests using VBA on Excel spreadsheet are shown in Table 3.6 shown below.

Table 3.6 Performance of VBA on Excel

		VBA Performance (seconds)			
		Total	Read	Algorithm	Write
Sort	Small size	0.0135	0.0002	0.0078	0.0055
	Medium size	0.1050	0.0034	0.0864	0.0152
	Large size	0.5496	0.0177	0.4745	0.0574
Shortest Path	Small size	0.2911	0.0117	0.0404	0.2390
	Medium size	1.0219	0.0254	0.1705	0.8260
	Large size	8.1588	0.2246	6.2037	1.7305
TSP	Small size	0.9426	0.0055	0.9273	0.0098
	Medium size	20.4383	0.0086	20.4145	0.0152
	Large size	98.9141	0.0039	98.9023	0.0078

It can be seen that from Sort to TSP test, with the increase of algorithm complexity, when algorithm computing becomes more and more complicated, VBA's algorithm computing time will increase tremendously and it will always take longer time compared with reading and writing on Excel spreadsheet. Meanwhile, within each implementation, the algorithm computing time will increase significantly with the increase of problem size. As a result, the total running time of the implementation will become extremely long.

With the growth of problem size, when the data transferring becomes more and more intensive, the reading and writing time of VBA in different implementation tests increases insignificantly except in Shortest Path. This is because the data format in Sort and TSP tests makes VBA able to read and write data in a group, but in Shortest Path test, data can only be read and written Cell by Cell. Furthermore, the reading time will always outperform the writing time, and therefore the data transferring time of VBA will mostly be composed of writing results back to Excel spreadsheet.

In summary, VBA performs much better on reading and writing than algorithm computing. Moreover, VBA performs very well on implementation with easy problem instead of a complicated one. Also, reading and writing time will be influenced by whether data is read or

written by group or by cell, as reading and writing by group will show much better performance. The performance results demonstrate the strength of VBA on reading and writing and the weakness of VBA on algorithm computation. VBA will probably be a good choice for implementations on Excel which do not contain complicated or difficult algorithm computing tasks. However, comparison of the performances with other methods on Excel spreadsheet is required to show its performance differences.

3.3.2 Performance of VC++ on Excel

The running time results of implementation tests using VC++ on Excel spreadsheet are shown in the table below.

Table 3.7 Performance of VC++ on Excel

		VC++ Performance (seconds)			
		Total	Read	Algorithm	Write
Sort	Small size	0.2693	0.0093	0.0003	0.2597
	Medium size	2.6745	0.1107	0.0030	2.5608
	Large size	13.4682	0.8595	0.0160	12.5927
Shortest Path	Small size	0.6466	0.0176	0.0010	0.6280
	Medium size	1.5995	0.0408	0.0045	1.5542
	Large size	8.7816	0.3815	0.1026	8.2975
TSP	Small size	0.0720	0.0011	0.0154	0.0555
	Medium size	0.3900	0.0019	0.3118	0.0763
	Large size	1.6818	0.0024	1.5810	0.0984

It can be seen that with the increase of algorithm complexity, when algorithm computing becomes more and more complicated, VC++'s algorithm computing time will remain at fast speed. Meanwhile, within each implementation, the algorithm computing time will increase insignificantly with the increase of problem size.

With the growth of problem size, when the data transferring becomes more and more intensive, the reading and writing time of VC++ in different implementation tests increases significantly

except in the TSP test. The data transferring work including reading and writing will be relatively easy in the TSP test since problem size in TSP test is only 50 to 150. Also, the reading time will always outperform the writing time, which means the data transferring time of VC++ will mostly be contributed by writing results back to Excel spreadsheet.

Thus, through these implementation tests, VC++ reveals its better performance and strength on algorithm computing compared to reading and writing. VC++ will be a good choice for applications on Excel spreadsheet with complicated algorithm computing but will not be able to provide good performance for applications with intensive data transferring. Comparison with other methods will be needed to conclude its performance differences.

3.3.3 Performance of Java on Excel

The running time results of implementation tests using Java on Excel spreadsheet are shown in the table below.

Table 3.8 Performance of Java on Excel

		Java Performance (seconds)			
		Total	Read	Algorithm	Write
Sort	Small size	0.1647	0.0429	0.0002	0.1216
	Medium size	0.2813	0.0628	0.0024	0.2162
	Large size	1.1403	0.1927	0.0120	0.9357
Shortest Path	Small size	0.5437	0.1289	0.0074	0.4074
	Medium size	0.8240	0.1921	0.0228	0.6092
	Large size	3.2271	0.9331	0.6298	1.6643
TSP	Small size	0.2326	0.0384	0.0776	0.1166
	Medium size	1.7104	0.0382	1.5523	0.1199
	Large size	7.9047	0.0398	7.7448	0.1201

It can be seen that with the increase of algorithm complexity, when algorithm computing becomes more and more complicated, Java’s algorithm computing time will remain to be short.

However, within each implementation, the algorithm computing time will grow quickly with the increase of problem size.

With the growth of problem size and when the data transferring becomes more and more intensive, the reading and writing time of Java in different implementation tests increases quickly. Moreover, the reading time will always outperform the writing time, which means the data transferring time of Java will mostly be from writing results back to Excel spreadsheet.

Hence, Java possesses the strength on algorithm computing compared to reading and writing on Excel spreadsheet. However, Java will not be able to provide very fast performance for applications with intensive data transferring or with very complicated algorithm computing. Its performance differences can be illustrated with the comparison to other methods on Excel spreadsheet.

3.3.4 Performance of VBA call C++ DLL on Excel

The running time results of implementation tests using VBA call C++ DLL on Excel spreadsheet are shown in Table 3.9 presented below.

Table 3.9 Performance of VBA call C++ DLL on Excel

		VBA call C++ DLL Performance (seconds)			
		Total	Read	Algorithm	Write
Sort	Small size	0.0064	0.0004	0.0005	0.0055
	Medium size	0.0246	0.0037	0.0050	0.0159
	Large size	0.1039	0.0193	0.0257	0.0588
Shortest Path	Small size	0.4089	0.0169	0.0057	0.3862
	Medium size	0.9002	0.0312	0.0215	0.8475
	Large size	3.0023	0.2358	0.7044	2.0621
TSP	Small size	0.0449	0.0008	0.0313	0.0129
	Medium size	0.3258	0.0012	0.3082	0.0164
	Large size	1.5754	0.0012	1.5602	0.0141

As demonstrated, the performance of VBA call C++ DLL on Excel spreadsheet combines the strength of VBA on data transferring and VC++ on algorithm computing. With the increase of

algorithm complexity, when algorithm computing becomes more and more complicated, VBA call C++ DLL's algorithm computing time increases but retains in fast speed. Within each implementation, the algorithm computing time will increase insignificantly with the increase of problem size.

With the growth of problem size and when the data transferring becomes more and more intensive, the reading and writing time of VBA call C++ DLL in different implementation tests remains to be very short except in the case of the Shortest Path test, since the data are read and written Cell by Cell in the Shortest Path implementation. Moreover, the data transferring time in VBA call C++ DLL is mainly caused by writing results back to Excel spreadsheet.

Thus, VBA call C++ DLL has the advantage on data transferring similar to VBA. Meanwhile, it also possesses the strength on algorithm computing similar to VC++ which is able to complete complicated algorithm computing tasks in a short period of time. Therefore, for applications on Excel spreadsheet both with intensive data transfer and with complicated algorithm computation, VBA call C++ DLL method will provide fast speed performance.

3.3.5 Comparison of Different Methods on Excel

In order to find out the performance differences between different methods on Excel spreadsheet, the best way is to compare the performance of different methods under the same criteria. From previous sections, we are able to tell the strengths and weaknesses of different methods on Excel spreadsheet. Also, we obtain the insight that data transferring time of different methods will be dominated by writing results back to Excel spreadsheet. Hence, we can combine the reading and writing performance, and compare the data transferring performance of different methods instead. Next, we will compare the performance, namely the running time, of these 4 methods on Excel in three aspects, including data transferring, algorithm computing and total time. These aspects are comprehensively examined with the increase of problem size and algorithm complexity.

(1) The Comparison of the data transferring (reading and writing) performance of four different methods on Excel spreadsheet is shown in Table 3.10 below.

Table 3.10 Performance comparison of different methods on Excel on data transferring

		Data Transferring Performance (seconds)			
		VBA	VC++	Java	VBA call C++ DLL
Sort	Small size	0.0057	0.269	0.1645	0.0059
	Medium size	0.0186	2.6715	0.279	0.0196
	Large size	0.0751	13.4522	1.1284	0.0781
Shortest Path	Small size	0.2507	0.6456	0.5363	0.4031
	Medium size	0.8514	1.595	0.8013	0.8787
	Large size	1.9551	8.679	2.5974	2.2979
TSP	Small size	0.0153	0.0566	0.155	0.0137
	Medium size	0.0238	0.0782	0.1581	0.0176
	Large size	0.0117	0.1008	0.1599	0.0153

It can be seen that the VBA and VBA call C++ DLL methods present great advantage on reading and writing, and provide the fastest performance on data transferring consistently. With the growth of data size, the data transferring time of VC++ and Java on Excel spreadsheet will increase quickly. Therefore, the order of the data transferring performance of these 4 methods on Excel spreadsheet is, VBA = VBA call C++ DLL > Java > VC++, where "=" means the same and ">" means faster than.

(2) The comparison of the Algorithm computing performance of four different methods on Excel spreadsheet is shown in Table 3.11 below.

As displayed, VC++ shows the fastest performance on algorithm computing consistently, and VBA also shows very fast speed on algorithm computing. However, its performance will be a little bit longer than VC++ as there is additional data passing time between VBA and C++ DLL. On the other hand, VBA always performs the slowest on algorithm computing. With the increase of the algorithm complexity, Java will be less efficient compared with VC++ on

algorithm computation. Hence, the algorithm computing performance difference of these 4 methods on Excel spreadsheet is VC++ > VBA call C++ DLL > Java > VBA, where ">" means faster than.

Table 3.11 Performance comparison of different methods on Excel on algorithm computing

		Algorithm Computing Performance (seconds)			
		VBA	VC++	Java	VBA call C++ DLL
Sort	Small size	0.0078	0.0003	0.0002	0.0005
	Medium size	0.0864	0.003	0.0024	0.005
	Large size	0.4745	0.016	0.012	0.0257
Shortest Path	Small size	0.0404	0.001	0.0074	0.0057
	Medium size	0.1705	0.0045	0.0228	0.0215
	Large size	6.2037	0.1026	0.6298	0.7044
TSP	Small size	0.9273	0.0154	0.0776	0.0313
	Medium size	20.4145	0.3118	1.5523	0.3082
	Large size	98.9023	1.581	7.7448	1.5602

(3) The comparison of the Total time performance of four different methods on Excel spreadsheet is shown in the table below.

Table 3.12 Performance comparison of different methods on Excel on Total time

		Total time Performance (seconds)			
		VBA	VC++	Java	VBA call C++ DLL
Sort	Small size	0.0135	0.2693	0.1647	0.0064
	Medium size	0.105	2.6745	0.2813	0.0246
	Large size	0.5496	13.4682	1.1403	0.1039
Shortest Path	Small size	0.2911	0.6466	0.5437	0.4089
	Medium size	1.0219	1.5995	0.824	0.9002
	Large size	8.1588	8.7816	3.2271	3.0023
TSP	Small size	0.9426	0.072	0.2326	0.0449
	Medium size	20.4383	0.39	1.7104	0.3258
	Large size	98.9141	1.6818	7.9047	1.5754

It can be seen that for the overall performance, VBA call C++ DLL will provide the fastest performance in total consistently, as it combines the advantage of VBA on data transferring and the strength of VC++ on algorithm computing. For other methods, VBA will beat VC++ and Java when there is intensive data transferring and the problem is simple and easy, such as the Sort test. However, as the problem becomes more and more complicated and data transferring becomes less and less intensive, VC++ and Java's advantage on algorithm computing will outperform and beat VBA completely, such as for the TSP test.

3.3.6 Summary

With the comprehensive performance comparison of different methods on Excel spreadsheet, we can summarize the performance differences of these four methods to build applications on spreadsheet.

For VBA on Excel spreadsheet, VBA has the fast speed in data transferring on Excel spreadsheet and the advantage of reading and writing data by groups. VBA is slow at algorithm computing and this weakness makes its performance the longest when the problem becomes more and more complicated. Hence, VBA, as an Internal programming method, will be fast for applications on Excel spreadsheet with intensive data transferring while slow for implementations with complicated algorithm computing.

For VC++ on Excel spreadsheet, VC++ has the fast speed on algorithm computing and this advantage makes it able to solve complicated problems and obtain the solution in very short time. VC++ is slow at data transferring on Excel spreadsheet and it will underperform when data transferring becomes more and more intensive. Thus, VC++, as an External programming method, will be fast for applications on Excel spreadsheet with complicated algorithm computing while slow for implementations with intensive data transferring.

For Java on Excel spreadsheet, Java also reveals its strength on algorithm computing and performs similarly as VC++ on Excel spreadsheet. However, Java's algorithm computing speed will not be as fast as VC++ with the increase of algorithm complexity. Meanwhile,

compared with VC++, Java reveals its strength in data transferring on Excel spreadsheet. Therefore, Java, as another External programming method, will be a good choice for applications on Excel spreadsheet with less intensive data transfers and less complicated algorithm computations.

For VBA call C++ DLL on Excel spreadsheet, this method combines the advantage of VBA on data transferring and the strength of C++ on algorithm computing. Hence, it reveals the fastest performance consistently through the implementation tests. However, there will be additional time for data exchange between VBA and C++ DLL file.

3.4 PERFORMANCE COMPARISON OF DIFFERENT METHODS ON CALC

Through the comprehensive implementation tests built on Calc spreadsheet using different methods, we are able to show the performance of different methods on Calc spreadsheet in the following sub-sections.

Again, for the different methods on Calc spreadsheet, two different methods are selected and their performances on Calc spreadsheet are compared. They are OOO Basic and Java. OOO Basic is an Internal programming method, and Java is an External programming method. They are the most typical methods to build applications on Calc spreadsheet. Hybrid programming method, to our best knowledge, will not be available as data is not stored consecutively in OOO Basic in Calc. Therefore, it is hard to combine OOO Basic with External programming methods as data is hard to exchange between Internal and External methods.

In order to tell the performance differences of different methods on Calc spreadsheet, the running times of different implementation tests on three aspects, data transferring, algorithm computing and Total time are compared comprehensively with increasing problem size and algorithm complexity.

In sub-sections 3.4.1 to 3.4.2, we will present the performance results of OOO Basic and Java on Excel. In sub-section 3.4.3, we will compare their performances. In sub-section 3.4.4, we will summarize their performance differences.

3.4.1 Performance of OOO Basic on Calc

The running time results of OOO Basic on Calc spreadsheet are shown in Table 3.13 presented below.

Table 3.13 Performance of OOO Basic on Calc

		OOO Basic Performance (seconds)			
		Total	Read	Algorithm	Write
Sort	Small size	7.347	0.749	3.915	2.683
	Medium size	15.257	1.373	8.252	5.632
	Large size	96.611	6.536	47.050	43.025
Shortest Path	Small size	14.826	2.624	8.374	3.828
	Medium size	57.642	10.155	34.570	12.917
	Large size	*Out of Memory*	*Out of Memory*	*Out of Memory*	*Out of Memory*
TSP	Small size	190.914	0.015	190.867	0.032
	Medium size	4791.056	0.047	4790.916	0.078
	Large size	24433.080	0.059	24432.955	0.093

Note that OOO Basic in the Shortest Path test with data size 5000 will be out of memory as 5000*5000 elements are too large in quantity to be accessed, and hence, the running time result is not available.

It can be seen that with the increase of algorithm complexity, when algorithm computing becomes more and more complicated, the algorithm computing time of OOO Basic increases tremendously and turns out to be extremely long in the TSP test. Meanwhile, with the growth of problem size, when the data transferring becomes more and more intensive, the data transferring time of OOO Basic, which includes reading and writing on Calc, also grows and becomes very long. Moreover, within data transferring, there is no dominance between reading

and writing. Overall, OOO Basic reveals better performance on data transferring than algorithm computing.

3.4.2 Performance of Java on Calc

To illustrate the performance of Java on Calc spreadsheet, the reading, algorithm computing, writing and total time results of Java in Sort, Shortest Path and TSP tests are presented in Table 3.14 shown below.

Table 3.14 Performance of Java on Calc

		Java Performance (seconds)			
		Total	Read	Algorithm	Write
Sort	Small size	5.562	2.591	0.001	2.970
	Medium size	588.166	292.811	0.016	295.338
	Large size	13292.494	6587.224	0.127	6705.144
Shortest Path	Small size	16.160	7.501	0.006	8.653
	Medium size	72.234	28.445	0.022	43.767
	Large size	2888.850	2539.089	0.566	349.196
TSP	Small size	0.584	0.056	0.257	0.270
	Medium size	6.647	0.114	6.159	0.373
	Large size	27.903	0.204	27.166	0.532

It can be seen that the Java method on Calc spreadsheet shows great strength on algorithm computing. However, the data transferring speed of Java, including reading and writing on Calc spreadsheet, is very slow. Except that in TSP test, Java's reading and writing time on spreadsheet is very short since the data size in TSP test is only 50 to 150. With the increase of problem size, the reading and writing time will increase and becomes extremely long. With the growth of algorithm complexity, the algorithm computing will remain to be within reasonable time. Moreover, no dominance between reading and writing exists within data transferring. In summary, Java performs at fast speed on algorithm computing while it reveals slow performance on data transferring on Calc spreadsheet.

3.4.3 Comparison of Different Methods on Calc

In order to find out the performance differences between different methods on Calc spreadsheet, we need to compare the performance of these two methods in different implementation tests presented above. The running time results of OOO Basic and Java for different implementation tests can be compared on three aspects, including data transferring, algorithm computing and total time, in which reading and writing are combined into data transferring. Through a comprehensive comparison with increasing problem size and algorithm complexity, the performance differences of different methods can then be shown.

(1) The Comparison of the data transferring performance of different methods on Calc spreadsheet is shown in Table 3.15 presented below.

Table 3.15 Performance comparison of different methods on Calc on data transferring

		Data transferring Performance (seconds)	
		OOO Basic	Java
Sort	Small size	3.432	5.561
	Medium size	7.005	588.149
	Large size	49.561	13292.37
Shortest Path	Small size	6.452	16.154
	Medium size	23.072	72.212
	Large size	*Out of Memory*	2888.285
TSP	Small size	0.047	0.326
	Medium size	0.125	0.487
	Large size	0.152	0.736

It can be seen that compared with Java, OOO Basic reveals its great strength in data transferring on Calc spreadsheet, and the data transferring performance of OOO Basic is consistently better than Java in all implementation tests.

(2) The comparison of the algorithm computing performance of different methods on Calc spreadsheet is shown in Table 3.16 below.

It can be seen that Java has great advantage on algorithm computing and performs at much faster speed than OOO Basic.

Table 3.16 Performance comparison of different methods on Calc on algorithm computing

		Algorithm Computing Performance (seconds)	
		OOO Basic	Java
Sort	Small size	3.915	0.001
	Medium size	8.252	0.016
	Large size	47.05	0.127
Shortest Path	Small size	8.3742	0.006
	Medium size	34.57	0.022
	Large size	*Out of Memory*	0.566
TSP	Small size	190.867	0.257
	Medium size	4790.916	6.159
	Large size	24432.955	27.166

(3) The comparison of the total time performance of different methods on Calc spreadsheet is shown in the table below.

Table 3.17 Performance comparison of different methods on Calc on Total time

		Total time Performance (seconds)	
		OOO Basic	Java
Sort	Small size	7.347	5.562
	Medium size	15.257	588.166
	Large size	96.611	13292.494
Shortest Path	Small size	14.8262	16.160
	Medium size	57.642	72.234
	Large size	*Out of Memory*	2888.850
TSP	Small size	190.914	0.584
	Medium size	4791.056	6.647
	Large size	24433.080	27.903

It can be seen that when the data transferring is very intensive, such as in Sort test, OOO Basic will show better performance than Java on Calc spreadsheet due to its strength on data transferring. When the problem becomes more and more complicated and data transferring

becomes less and less intensive, which means that the algorithm computing will play the most important role, Java can make use of its strength on algorithm computing and outperform OOO Basic in Total on Calc spreadsheet. Furthermore, when the algorithm computing and data transferring are both at a medium level, these two methods will provide similar performance.

3.4.4 Summary

Through the performance comparison of OOO Basic and Java on Calc spreadsheet, the performance differences of different methods on Calc spreadsheet can be summarized below.

For OOO Basic on Calc spreadsheet, OOO Basic has the strength of data transferring on Calc spreadsheet. However, OOO Basic performs very slowly on algorithm computing. Hence, OOO Basic, as an Internal programming method to build applications on Calc spreadsheet, will be appropriate for applications with intensive data transferring while it will be slow for applications with complicated algorithm computing. In addition, OOO Basic is not able to access very large dimensional matrices which limit its scope of use.

For Java on Calc spreadsheet, Java has fast speed on algorithm computing, but it is slow at data transferring on Calc spreadsheet. This combination of performance determines that Java, as an External programming method, will be a good choice for applications on Calc spreadsheet with complicated algorithm computing but easy data transferring work.

3.5 EASE OF IMPLEMENTATION OF DIFFERENT METHODS ON SPREADSHEET

After determining the performance differences of different methods to build computational applications on spreadsheet, the next natural question will be the ease of implementation of different methods to build spreadsheet applications. This is because if the performance among various methods turns to be indifferent in certain scenarios, people will certainly want to select the one with the least implementation effort.

Therefore, in this section, we intend to conduct the ease of implementation analysis of different methods on spreadsheet to help people to tell their implementation effort differences. Thus people are able to select the most efficient method to build spreadsheet applications that can achieve the performance requirement while implement with the least effort.

To analyze the ease of implementation, we assume that the implementation effort required can be revealed in terms of the code structures and the codes to be written for unsophisticated developers or inexperienced programmers, since more critical codes always refer to more implementation effort when building computational applications on spreadsheet. Hence, through the analysis of the essential codes needed and the code structures to build applications on spreadsheet using each method, the ease of implementation of different methods on spreadsheet can be shown.

To build applications on spreadsheet, the most important effort to be implemented is the interface between different methods and the spreadsheet to transfer Input data and Output result, since the implementation effort for algorithm computing part will be about the same for every method to obtain the solution. Thus, in sub-sections 3.5.1 to 3.5.6, we will illustrate the critical codes of different methods to build up the interface with spreadsheet intuitively. In sub-section 3.5.7, we conclude on the ease of implementation of different methods on spreadsheet.

3.5.1 Implementation of VBA to Build Applications on Excel

VBA is an application of Microsoft's event-driven programming language Visual Basic and its associated Integrated Development Environment (IDE), which are built into Microsoft Excel spreadsheet. VBA enables developers to build self-defined functions and applications. Hence, we can use VBA to build applications on Excel spreadsheet directly.

In order to build applications on Excel spreadsheet with VBA, we need to build up the interface between VBA and Excel spreadsheet, which contains three level of access on Excel: *Workbook*, *Worksheets*, and *Cells* or *Range*. These three levels form a relation of layers, which means that lower levels can only be accessed if we have obtained the access of upper level. For

instance, if we want to read and write the data on Excel spreadsheet, we have to first access the *Workbook*, then the *Worksheets*, before we can access the *Cells* to read and write data.

Hence, the interface between VBA and Excel spreadsheet requires the 3 steps shown below:

Step 1. Obtain the access of Workbook layer. In VBA, we can use the Application's property and function, and declare (Dim) a *Workbook* type variable to obtain the access of Workbook level as shown in the figure below.

```
Option Explicit
Sub main()

Dim work_book As Workbook
Set work_book = Application.Workbooks.Open("d:\test.xls", True, True)
work_book.Activate
```

Figure 3.1 Access Excel Workbook layer with VBA

Step 2. Obtain the access of Worksheet layer. After Workbook, we can use the *Workbook*'s property and function, and declare (Dim) a *Worksheet* type variable to obtain the access of Worksheet level as shown in the figure below.

```
Dim work_sheet As Worksheet
Set work_sheet = work_book.Worksheets("Sheet1")
```

Figure 3.2 Access Excel Worksheet layer with VBA

Step 3. Obtain the access of Cells level. After Worksheet, we can access the *Cells* by using the *Worksheet* variable's property function as shown in the figure below.

```
Dim data(n) As Integer
Dim i As Integer
For i = 1 To n
    data(i) = work_sheet.Cells(i, 1).Value
Next i
```

Figure 3.3 Access Excel Cells layer with VBA

Here, *i* stands for row index and *j* stands for column index in *Cells(i, j)*. The Value property will return the data information stored in this cell.

More detailed information of routines and sample codes of VBA on Excel spreadsheet can be found in APPENDIX A.

3.5.2 Implementation of VC++ to Build Applications on Excel

With the purpose of building applications on Excel spreadsheet with VC++, the most important part is to build up the interface between VC++ and Excel spreadsheet. In this research, we will apply a VC++ open-source library written by Yap (2006) to read and write data on Excel spreadsheet. This open-source library called BasicExcel is a C++ class that is able to bridge VC++ with Excel 2003 spreadsheet. The way that BasicExcel interfaces with Excel is to operate the file directly through the Binary file format. Microsoft Excel uses a proprietary binary file format called BIFF (Binary Interchange File Format) as its primary format up until Excel 2007, and since version 2007, Microsoft Excel changes its file format to Office Open XML. Therefore, BasicExcel is restricted to be used for Excel with version earlier than 2007.

Basic reading and writing tasks on 2003 Excel spreadsheet are supported by this open-source library, such as reading and writing numbers and strings, adding and deleting worksheets, geting name of or renaming the worksheets. Although it looks simple, BasicExcel has already satisfied all the basic steps that are to be covered in the implementation tests in this research.

BasicExcel contains three Classes to approach three layers of access on Excel spreadsheet. They are Class *BasicExcel* for Workbooks, Class *BasicExcelWorksheet* for Worksheets and Class *BasicExcelCell* for Cells. The functions used in implementation tests and their descriptions are listed in the table below.

Table 3.18 Functions of C++ BasicExcel Library to access Excel

Functions in BasicExcel	Description
bool Load (Const char* filename)	Load a workbook from an Excel spreadsheet file.
bool SaveAs (Const char* filename)	Save current workbook to an Excel file.

BasicExcelWorksheet* GetWorksheet(size_t sheetIndex)	Generate a pointer to an Excel worksheet with given index. Index starts from 0.
BasicExcelCell* Cell(size_t row, size_t col)	Give a pointer to an Excel cell. Row index and Col index start from 0.
double GetInteger() const	Return an integer value in Cell.
void SetInteger(int val)	Set content of an Excel cell to an integer value.
double GetDouble() const	Return a double value in Cell.
void SetDouble(double val)	Set content of an Excel cell to a double value.

The interface between VC++ and Excel spreadsheet can be built through the 3 steps shown below:

Step 1. Load Workbook file. The access of Workbook can be obtained by calling the *"Load()"* function in Class "*BasicExcel*" as shown in the figure below.

```
BasicExcel work_book;
work_book.Load("D:\\test.xls");
```

Figure 3.4 Access Excel Workbook layer with C++ Library

Step 2. Access Worksheet. The access of Worksheet can be obtained by calling the "*GetWorksheet()*" function in Class "*BasicExcel*" and passing to a pointer declared as "*BasicExcelWorksheet*" object as shown in the figure below.

```
BasicExcelWorksheet* work_sheet = work_book.GetWorksheet("Sheet1");
```

Figure 3.5 Access Excel Worksheet layer with C++ Library

Step 3. Access Cells. The access of Cells can obtained by calling the function "*Cell(i, j)*" in Class "*BasicExcelWorksheet*" and received by declaring a pointer to "*BasicExcelCell*" object as shown in Figure 3.6 presented below.

Here, *i* stands for row index and *j* stands for column index in *Cell(i, j)*. The row index and column index start from 0.

```
int* data=new int [n];
for (size_t i=0; i<n; ++i)
{
   BasicExcelCell* cell =work_sheet->Cell(i,0);
   data[i]=cell->GetInteger();
}
```

Figure 3.6 Access Excel Cells layer with C++ Library

More sample codes of how to interface with Excel spreadsheet using the VC++ BasicExcel library method and more detailed routine information can be found in APPENDIX A.

3.5.3 Implementation of Java to Build Applications on Excel

In this research framework, we will use a Java open-source library called JXL to interface with Excel and manipulate the Excel spreadsheet. JXL is a mature Java API (Application Programming Interface) that allows people to read and write on Excel spreadsheet. Within its features, JXL supports reading and writing data on Excel 2003 spreadsheets. It also supports basic operations such as setting format, adding and removing worksheets and so on. In summary, JXL is a maturely developed, free Java Excel API popularly used to build up the interface between Java and Excel spreadsheet nowadays.

Similarly, JXL can access three layers of Excel spreadsheet using the Class methods and interfaces inside the library package. JXL divides reading and writing on Excel spreadsheet to two different Class packages, where "jxl" is dedicated to handle reading tasks and "jxl.write" is dedicated to handle writing tasks. For reading Workbook, a Class called "*Workbook*" is defined in "jxl" to represent a Workbook and provide a handle into individual Worksheets. For Worksheet, a public interface called "*Sheet*" is defined to represent a Worksheet within a Workbook and provide a handle into individual Cells. For Cells, a public interface called "*Cell*" is defined to represent an individual Cell within a Worksheet and can be queried for its type and its contents. Within the interfaces Cell, there are also sub-interfaces, such as *NumberCell*, *FormulaCell*, etc. to generate different types of Cells. Writing will be very similar to reading, and the difference is to replace the above Classes and interfaces to *WritableWorkbook*, *WritableSheet*, and *WritableCell*.

The Class methods used in implementation tests to interface with Excel spreadsheet, and their descriptions are listed in Table 3.19 presented below.

Table 3.19 Methods of Java JXL Library to access Excel

Commonly used Methods in JXL	Description
public static Workbook getWorkbook (java file)	A factory method which takes in an Excel file
public abstract Sheet getSheet (int i)	Gets the Worksheet within this workbook with specified Index
public Cell getCell (int column, int row)	Returns the cell specified at this row and column
Sheet.public int getRows()	Returns the number of rows in this sheet
NumberCell getValue()	Gets the double contents for this cell
public static WritableWorkbook createWorkbook (java file)	Creates a writable workbook with the given file name
public WritableSheet getSheet (int i)	Gets the specified sheet within this workbook
public void addCell (WritableCell cell)	Adds a cell to this sheet
public class Number	Creates a cell, which contains a numerical value
public Number(int col, int row, double val)	Constructs a number, adds to a spreadsheet at the column/row position indicated.
public abstract void write()	Writes out the data held in this workbook in Excel format

The interface between Java and Excel spreadsheet can be built through the 3 steps shown below:

Step 1. Access the Workbook. The access of Workbook can be obtained by calling the *getWorkbook()* function and passing to the Class *Workbook* objects shown in Figure 3.7 shown below.

```
Workbook workbook=null;
try
{
    workbook=Workbook.getWorkbook(new File(path));
}
catch(Exception e)
{
    System.out.println(e);
}
```

Figure 3.7 Access Excel Workbook layer with Java Library

Step 2. Access the Worksheet. The access of Worksheet can be obtained by calling the *getSheet()* function and passing to the Class *Sheet* object as shown in the figure below.

```
Sheet work_sheet=workbook.getSheet(0);
```

Figure 3.8 Access Excel Worksheet layer with Java Library

Step 3. Access the Cells. The access of Cells can be obtained by calling the *getCell(i, j)* function and passing to the Class *NumberCell* object. The value can be retrieved by calling the *getValue()* function as shown in the figure below.

```
int n=work_sheet.getRows();
int data[]=new int [n];
NumberCell cell=null;
for (int i=0;i<n;i++)
{
    NumberCell numcell=(NumberCell)work_sheet.getCell(0,i);
    data[i]=(int)numcell.getValue();
}
```

Figure 3.9 Access Excel Cells layer with Java Library

Here, *i* stands for column index and *j* stands for row index in *getCell(i, j)*. The row index and column index start from 0.

More sample codes and detailed information of Java routines on Excel can be found in APPENDIX A.

3.5.4 Implementation of VBA call C++ DLL to Build Applications on Excel

Dynamic-Link library, or DLL, is Microsoft's implementation of the shared library concept in the Microsoft Windows systems. DLL files are independent modules that contain functions and resources which are compiled, linked and stored separately from the applications. It provides a way to modularize applications so that functionality can be updated, reused, and most importantly, be shared. Hence, this property allows other applications to call the DLL file as a function. In this research framework, once the DLL is defined and the format is matched, VBA is able to call DLL as a function to perform execution of algorithms and specific computations to obtain the results. We will use C++ to compile the DLL function and declare

it in VBA. This can thus be considered as a Hybrid programming language since it is the combination of VBA and C++ programming languages.

The process flow of the implementation is that we first read data with VBA from Excel spreadsheet, then call the DLL function, and the input data are transferred into C++ DLL to carry out algorithm computations and results are obtained. Thereafter, the output results are transferred from DLL back to VBA, and VBA will write the results back to Excel spreadsheet.

We can see that to build applications on Excel spreadsheet using VBA call C++ DLL method, two kinds of interface have to be built. One is the bridge linking Excel spreadsheet with VBA, and the other one is the bridge linking VBA with C++ DLLs. The interface with Excel spreadsheet follows the same reading and writing procedure described in VBA method on Excel, which contains three layers of access to Excel spreadsheet through VBA. Hence, we have to build up the bridging interface between VBA and C++ DLL to transfer the data and results.

In order to build up the interface between VBA and C++ DLL, function arguments are used to transfer input data and output results. The important information to be noted is that the data type length of function parameters must be matched between C++ DLL and VBA. The data types commonly used in C++ DLLs and their VBA equivalents are shown in the table below.

Table 3.20 The Equivalents of common data types between C++ DLL and VBA

C++	VBA	Size in Bytes	Description
bool	Boolean	2	Stores a value of True (0) or False (-1)
short	Integer	2	Contains a number in the range of -32768 to 32767
int	Long	4	Contains a number in the range of -2,147,483,648 to 2,147,483,647
double	Double	8	Contains a real number in the range of +/- 1.7E +/- 308
char	Byte	1	Contains a number in the range of 0-255

Since the implementation codes of VBA call C++ DLL method on Excel spreadsheet are long and complicated, the detailed routine and sample codes of VBA call C++ DLL method on Excel are provided in APPENDIX A.

3.5.5 Implementation of OOO Basic to Build Applications on Calc

OOO Basic is a programming language developed especially for OpenOffice.org applications and it is integrated into the OpenOffice.org package. OOO Basic belongs to the Basic family, and it is very similar to VBA for Excel spreadsheet. We can insert modules in the OOO Basic editor and build applications on Calc spreadsheet directly.

Similarly, in order to use OOO Basic to operate on Calc spreadsheet, we also need to build up the interface between OOO Basic and Calc spreadsheet which contains three levels of access on Calc: *SpreadsheetDocument*, *Sheets*, and *Cells* or *Range*. Hence, the interface between OOO Basic and Calc spreadsheet requires the 3 steps shown below.

Step 1. Obtain the access of *SpreadsheetDocument*. The access of *SpreadsheetDocument* can be obtained by calling the "*LoadComponentFromURL()*" function and declaring an object type variable to receive it. If the spreadsheet document has already been opened and the application is built directly on it, then we can simply declare an object and use "*ThisComponent*" to obtain the access of this *SpreadsheetDocument* as shown in the figures below.

```
Dim doc as Object
Dim Property()
doc=starDesktop.LoadComponentFromURL("file:///D:/test.odt","_blank", 0, Property)
```

Figure 3.10 Access Calc SpreadsheetDocument layer with OOO Basic

```
Dim doc as Object
doc=ThisComponent
```

Figure 3.11 Access opened Calc SpreadsheetDocument directly with OOO Basic

Step 2. Obtain the access of *Sheets*. The access of *Sheets* can be obtained by calling the "*Sheets()*" function and received by declaring an object type variable as shown in the figure below.

```
Dim sheet as Object
sheet=doc.Sheets(0)
or
sheet=doc.Sheets.getByName("Sheet1")
```

Figure 3.12 Access Calc Sheets layer with OOO Basic

Step 3. Obtain the access of *Cells*. The access of *Cells* can be specified by calling the "*getCellByPosition(i, j)*" function and received by declaring an object type variable, where *i* stands for the column index and *j* stands for the row index starting from 0, as shown in the figure below.

```
Dim i as Integer
Dim cell as Object
Dim data(n) as Long
For i = 0 to n
   cell=sheet.getCellByPosition(0,i)
   data(i)=cell.value
Next i
```

Figure 3.13 Access Calc Cells layer with OOO Basic

More detailed OOO Basic routines and sample codes can be found in APPENDIX B.

3.5.6 Implementation of Java to Build Applications on Calc

In this research framework, we will use a free Java library called ODFDOM toolkit to build up the interface and access Calc spreadsheet. ODFDOM is an Open Document API that provides an easy way to manipulate ODF (Open Document Format) files such as Calc spreadsheets. Moreover, based on ODFDOM, we also apply a more easy-to-use Simple Java API, which is a high level abstraction of the lower-level ODFDOM API for modifying ODF files. In order to use Simple Java API, the following runtime libraries are required:

- JDK version 1.6
- ODFDOM 0.8.7
- The Apache Xerces 2.9.1 or higher version

Just like other methods, Simple Java API also provides three layers of Classes to access Calc spreadsheet. For *SpreadsheetDocument*, a Class called "*SpreadsheetDocument*" is defined in

Simple Java API to represent a *SpreadsheetDocument*. For *Sheets*, a public Class called "*Table*" is defined to represent the *Sheets* in ODF spreadsheet and provide methods to modify cells. For *Cells*, a public Class "*Cell*" is defined to represent the *Cells* in ODF spreadsheet and provide methods to modify the cell content and cell properties. The Class methods used in implementation tests and their descriptions are listed in the table below.

Table 3.21 Methods of Java SimpleJavaAPI Library to Access Calc

Methods in Simple Java API on Calc	Description
public static SpreadsheetDocument.loadDocument (DocumentPath)	Loads a SpreadsheetDocument from the provided path
public Table getSheetByIndex (int i)	Retrieves sheet by index
public Cell getCellByPosition (int col, int row)	Returns a single cell that is positioned at the specified column and row
Class Cell. getDoubleValue ()	Gets the double value of this cell
Class Cell.setDoubleValue (Double value)	Sets the cell value as a double
public void save (OutputPath)	Saves the document to an OutputStream

The interface between Java and Calc spreadsheet can be built through the 3 steps shown below.

Step 1. Access the *SpreadsheetDocument*. The access of *SpreadsheetDocument* can be obtained by calling the "*loadDocument()*" function and passing to an object with "*SpreadsheetDocument*" Class type as shown in the figure below.

```
SpreadsheetDocument workbook =null;
try
{
    workbook=SpreadsheetDocument.loadDocument("D:\\test.xls");
}
catch(Exception e)
{
    System.out.println(e);
}
```

Figure 3.14 Access Calc SpreadsheetDocument layer with Java Library

Step 2. Access the *Sheets*. The access of *Sheets* can be obtained by using the *SpreadsheetDocument* object to call the "*getSheetByIndex()*" function to retrieve *Sheets* by index and passing to an object with "*Table*" Class type as shown in the figure below.

```
Table sheet=workbook.getSheetByIndex(0);
```

Figure 3.15 Access Calc Sheets layer with Java Library

Step 3. Access the *Cells*. The access of *Cells* can be obtained by using the *Table* object "*sheet*" defined above to call the "*getCellByPosition(i, j)*" function, where *i* stands for the column index and *j* stands for the row index starting from 0, And then passing to an object defined with "*Cell*" Class type as shown in the figure below.

```
Cell cell=null;
double data[]=new double [n];
for (int i=0; i<n; i++)
{
    cell=sheet.getCellByPosition(0,i);
    data[i]=cell.getDoubleValue();
}
```

Figure 3.16 Access Calc Cells layer with Java Library

More detailed information of the Java Routines on Calc spreadsheet and sample codes are shown in APPENDIX B.

3.5.7 Summary of Ease of Implementation of Different Methods on Spreadsheet

Through the illustration of implementation of different methods and the critical codes to construct the interface with spreadsheet, we are able to intuitively summarize the code structures of different methods to build applications on spreadsheet, as illustrated in Figure 3.17 below.

It can be seen that for Internal programming methods, such as VBA on Excel and OOO Basic on Calc, we can build up the interface directly on spreadsheet; for External programming methods, such as VC++, Java on Excel and Java on Calc, we have to import the open-source library and build up the interface with spreadsheet; for Hybrid programming methods, such as

VBA call VC++ DLL, we have to build up two interfaces, namely the interface to transfer data between spreadsheet and Internal method, and the interface to transfer data between Internal method and External method.

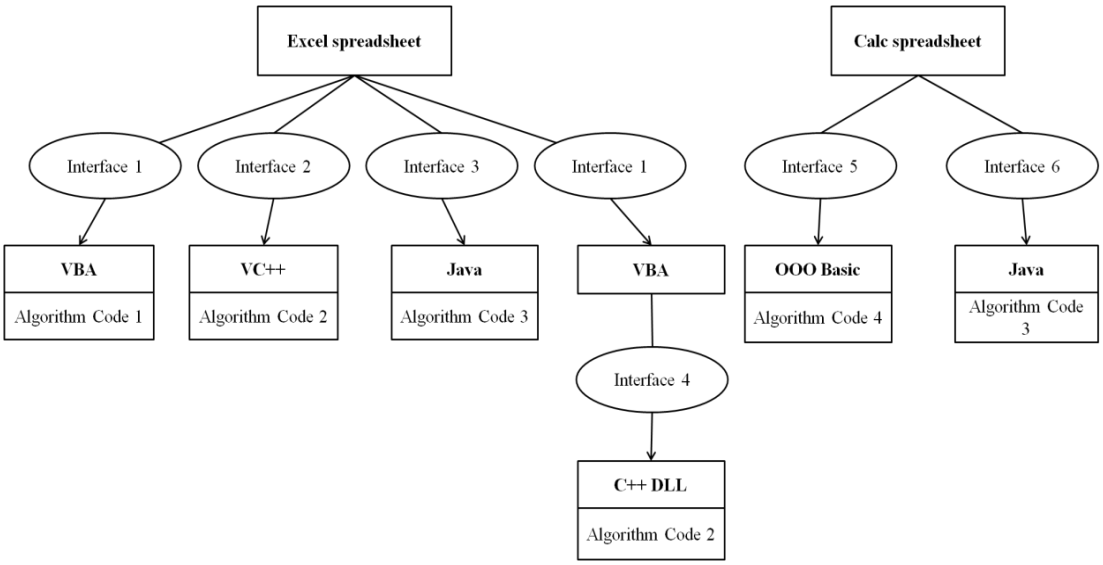

Figure 3.17 Code structures of different methods to build applications on spreadsheet

Therefore, through the codes structures and the critical codes to be written to build spreadsheet applications, the ease of implementation of different kinds of methods can be concluded as follows: Internal methods require the easiest effort to implement, then the External methods, and Hybrid methods require the most effort to implement to build applications on spreadsheet.

3.6 CONCLUSIONS

In this chapter, comprehensive implementation tests are conducted to show the performance differences and implementation effort of different methods to build applications on spreadsheet. Until now, we are able to answer the research questions 1 and 2 stated in Chapter 1. Conclusions are summarized below:

1. For the performance of different methods, VC++ and Java, as External programming methods, have fast speed on algorithm computing but suffer from the weakness in data transferring on spreadsheet. VBA and OOO Basic, which are Internal programming methods integrated inside Excel and Calc spreadsheet, perform at fast speed on data transferring, but

have weakness on algorithm computing. The Hybrid programming method, such as VBA call C++ DLL, which combines the advantages of Internal and External programming methods, will provide the overall fastest performance in both data transferring and algorithm computing.

2. For the ease of implementation, Internal methods require the least effort to implement, followed by External methods. Hybrid methods require the most effort to implement to build applications on spreadsheet.

Through various implementation tests, it can be seen that Hybrid programming methods have the great advantage in building computational spreadsheet applications. Although Hybrid methods will require more implementation effort than Internal and External programming methods, however, it is much more capable than Internal and External programming methods on spreadsheet in terms of speed performance, especially for spreadsheet applications to solve very complicated problems with sophisticated algorithms. We will elaborate on this issue in the next chapter.

Chapter 4

An Application Example: Solving VRPTW on Excel

4.1 INTRODUCTION

In Chapter 3, the performance differences between various options to build computational spreadsheet applications and their ease of implementation have been investigated. It is found that the Hybrid programming method, such as VBA call C++ DLL on Excel spreadsheet, shows the best performance consistently throughout the comparison, and hence it shows strong capability to build computational spreadsheet applications. However, it remains to investigate the capability of using this method to build spreadsheet applications to solve very complicated problems with sophisticated algorithms, and evaluate its performance in terms of speed compared with other standalone applications.

To investigate the capability of the Hybrid programming method, VBA call C++ DLL, we implement this method and build an Excel VRPTW application to solve VRPTW (Vehicle Routing Problem with Time Windows) problems. VRP is a NP-hard problem, which means that generally, optimal solutions cannot be obtained in Polynomial time. It is much more complicated than TSP as TSP is just a special case of VRP, and it is likely that the worst case running time of VRP increases exponentially with problem size. VRPTW is even harder than VRP as it has to consider the time window constraint. Spreadsheets have been used for solving various kinds of optimization problems (Parlar 1986, Roy, A., Lasdon and Plane 1989, Conway and Ragsdale 1997, Kharab 2000). However, these problems addressed are not as complicated as VRP.

We apply the tabu-search heuristics (Lau et al. 2003) to solve the VRPTW problem, and we use this Excel VRPTW application to solve all the Solomon test cases, which are well-established benchmark test cases for VRPTW problem (Solomon, 1987), and compare the performance with a standalone C++ application on solving this VRPTW problem. The

standalone C++ application will read and write data on text files. With the Excel VRPTW application example to solve a very complicated problem with sophisticated heuristics and the evaluation of its performance through the comparison with other standalone applications, we are able to tell the capability of using VBA call C++ DLL to build computational spreadsheet applications.

VBA call C++ DLL method has an inherent limitation of transferring dynamic length array data between VBA and C++ DLL. Here, we propose a Sync concept to overcome this limitation and we manage to transfer the VRPTW solution with dynamic length route result array from C++ DLL to VBA.

This Chapter is organized as follows: in section 4.2, we introduce the Excel VRPTW application example in detail, including the data format and the interface between VBA and C++ DLL. In section 4.3, we present the performance result of this VRPTW spreadsheet application. We conclude and summarize this Chapter in section 4.4.

4.2 EXCEL VRPTW APPLICATION USING VBA CALL C++ DLL METHOD

This VRPTW application example follows the same process as we discussed previously in using VBA call C++ DLL method to build spreadsheet applications. Firstly, Input data will be read from the Excel spreadsheet using VBA. Then Input data are transferred from VBA to C++ compiled DLL to solve the problem and the result will be obtained. After that, solution results are transferred back to VBA from C++ DLL. Finally, Output results are written back to the Excel spreadsheet using VBA. All the I/O data transferring occurs in memory, and the process can be illustrated in the figure below.

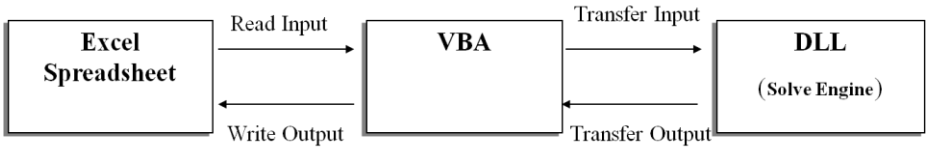

Figure 4.1 Data flow of spreadsheet applications built with VBA call C++ DLL method

4.2.1 Input and Output Format

The Input data format follows the same structure as in the Solomon test cases (Solomon, 1987), in which *Number of Vehicles*, *Vehicles Capacity Limit*, *Number of Customers*, *Customer Positions*, *Demand*, and *Time Window* are specified. The Input data format is shown in the figure below.

nNumberOfVehicles	25					
nCapacityLimit	1000					
nNumberOfCustomers	100					
Index	pCustomer PosX[i]	pCustomer PosY[i]	pCustomer Demand[i]	pCustomer ReadyTime[i]	pCustomer LateTime[i]	pCustomer ServiceTime[i]
0	35	35	0	0	1000	0
1	41	49	10	0	974	10
2	35	17	7	0	972	10

Figure 4.2 Iutput format of Excel VRPTW application

The Output will present the detailed information of the solution. Firstly, there will be the total summary information, including *Number of Vehicles Used*, *Number of Customers Served*, *Number of Customers Not Served*, *Total Distance Travelled*, and *Total Load Carried*. Then, the specific route information of each used vehicle will be shown, such as the *Vehicle ID*, *Number of Customers Served* in the route, *Customer ID* served in sequence, *Arrival Time*, and the *Start time* and *Due Time* of the customer. A sample of Output format is shown in Figure 4.3 below.

Route Report			
Number of vehicles used:	10		
Number of customers served:	100		
Number of customers not served:	0		
Total Distance Travelled:	829.01		
Total Load Carried:	1810		
Vehicle ID:	1		
Number of customers served:	10		
Customer	Arrival	Start	Due
90	2062	2000	8400
87	11562	8500	14400
86	20662	17300	23800
83	30262	26500	33800
82	39562	36900	42000
84	49145	45800	52300
85	58428	55500	61200
88	67728	64500	70800
89	77011	73700	80200
91	86372	83600	88900

Figure 4.3 Output format of Excel VRPTW application

4.2.2 Using VBA call C++ DLL to Build the Excel VRPTW Application

The purpose of using VBA call C++ DLL method to build applications on spreadsheet is to utilize the strength of VBA on performing data transferring tasks and also the advantage of C++ on algorithm computing. In order to use this method, we have to build up the interface to bridge VBA with C++ compiled DLL to transfer data between them.

DLL as a Dynamic Link Library is constructed by different functions fulfilling different tasks, and DLL is able to *EXPORT* these defined functions in this library. Accordingly, VBA is able to declare these exported functions in DLL. After declaring the functions, VBA can call these functions as self-defined functions, and then VBA is linked with DLL.

However, we have to make this bridge able to transfer data between each other. As illustrated in Chapter 3 and APPENDIX A, VBA has two ways to pass values: one is by value (*ByVal*), and the other is by reference or address (*ByRef*). *ByVal* is always used to pass single value, and *ByRef* is usually used to pass arrays or matrices. We will use function argument variables to transfer the data between VBA and C++ DLL, and we need to make sure the argument type and return type in the DLL exported function are uniformly matched with the ones in the VBA declared function.

For example, in this VRPTW application, we first compile a DLL file called "*VRP.dll*" to carry out the algorithm computation. Within this "*VRP.dll*", we define an Interface function called *VRPprocess* to export. In this interface function, the function arguments contain all the Input and Output variables. Hence the Input data and Output result will be exchanged between VBA and C++ DLL through these arguments, as shown in Figure 4.4 below.

It can be seen that the interface function *VRPprocess* will first transfer the Input from VBA to C++ DLL through the Input arguments, as done by the sub-function *TransferInput()*. Then the results can be obtained from *Optimise()* which performs the algorithm computing, and these results are passed from C++ DLL back to VBA through the Output arguments, as done by the sub-function *ReportOutput()*.

```
double _stdcall VRPprocess(int n_v, int n_caplimit, int n_customer, int * pPosX, int* pPosY, int* pDemand, \
                           int* pReadyTime, int* pLateTime, int* pServiceTime, int* n_VehiclesUsed, \
                           int* n_CustomerServed, int* n_CustomerNotServed, double* TotalDistance, \
                           int* TotalLoad, int* nCustomerInRoute, int* nCustomerNumber, int* nRouteArrivalTime)
{
   // Creat Class object;
   CVRPTW VRPTW;
   // Receive data from the argument variables
   VRPTW.TransferInput(n_v, n_caplimit, n_customer, pPosX, pPosY, pDemand, pReadyTime, \
                       pLateTime, pServiceTime);
   // Optimize and obtain the solution
   VRPTW.Optimise();
   // Transfer result to the argument variables
   VRPTW.ReportOutput(n_VehiclesUsed, n_CustomerServed, n_CustomerNotServed, TotalDistance, \
                       TotalLoad, nCustomerInRoute, nCustomerNumber, nRouteArrivalTime);
   // Free the memory and space
   VRPTW.OnFree();
   return 0;
}
```

Figure 4.4 Define Interface function *VRPprocess* in C++ DLL

More sample codes and routines of how to transfer data using function arguments can be found in APPENDICES A and C.

Then we create a "*VRP.def*" file to export this function, as shown in the figure of C++ sample code below.

```
LIBRARY VRP

EXPORTS
VRPprocess @1
```

Figure 4.5 Export *VRPprocess* function in C++ DLL

After that, we generate the DLL file and declare this function in VBA, as shown in the figure of VBA sample codes below.

```
Option Explicit

Public Declare Function VRPprocess Lib "C:\Documents\Visual Studio 2010\Projects\VRP\Release\VRP.dll" _
                                       (ByVal n_v As Long, ByVal n_caplimit As Long, ByVal n_customer As Long, _
                                        ByRef pPosX As Long, ByRef pPosY As Long, ByRef pDemand As Long, _
                                        ByRef pReadyTime As Long, ByRef pLateTime As Long, _
                                        ByRef pServiceTime As Long, ByRef n_VehiclesUsed As Long, _
                                        ByRef n_CustomerServed As Long, ByRef n_CustomerNotServed As Long, _
                                        ByRef TotalDistance As Double, ByRef TotalLoad As Long, _
                                        ByRef nCustomerInRoute As Long, ByRef nCustomerNumber As Long, _
                                        ByRef nRouteArrivalTime As Long) _
                                        As Double
```

Figure 4.6 Declare *VRPprocess* function in VBA

When declaring the exported C++ DLL function *VRPprocess()* in VBA, the function argument types are correspondingly matched, such as *int* data type in C++ matches *Long* data type in VBA, *int** pointer in C++ matches *ByRef as long* in VBA to pass the address of the variable value. The return type is also matched to be *Double* value for both. The common data type equivalence between C++ DLL and VBA is shown in Table 3.20 in Chapter 3.

Thus, we are able to transfer Input data and Output results between VBA and C++ DLL now. However, we can only pass static length array of data between VBA and C++ DLL. Since the array data is transferred using pointers by address, and calling the DLL function in VBA is a one-time trigger event, it is not able to dynamically change the length of array. Unlike TSP, the VRP result is the Route solution with dynamic length, and before algorithm computing, we have no chance to know the length of route result array in advance. Next, we propose the Sync concept in building the interface between VBA and C++ DLL to overcome such an inherent limitation of this method to build spreadsheet applications.

To illustrate intuitively, Figure 4.7 below shows the Sync process to transfer dynamic length array data between VBA and C++ DLL. The Sync concept is to transform the dynamic length array to a static length array that is large enough to handle the maximum dynamic length and the information of the length.

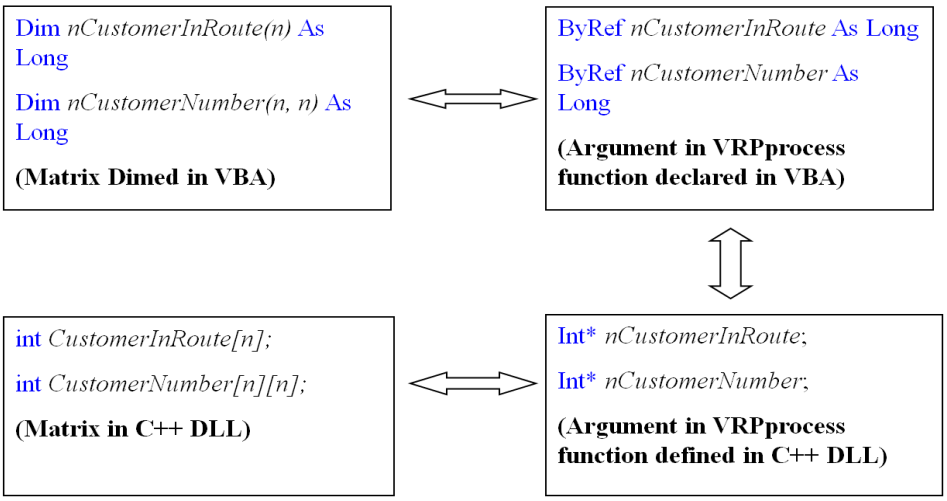

Figure 4.7 Sync process to transfer data array with dynamic length information between VBA and C++ DLL

It can be seen that the route result with *CustomerNumber* is a dynamic length result. For VBA and C++ DLL, we define the matrices *nCustomerNumber(n, n)* and *CustomerNumber[n][n]* that are both large enough to handle the maximum number of customers in the route result, and define the *nCustomerInRoute(n)* and *CustomerInRoute[n]* arrays to record the dynamic length information. After algorithm computation, the route solution result is stored in *CustomerNumber[n][n]* matrix and the number of customers in each route with dynamic length information is stored in *CustomerInRoute[n]*. Then through function arguments, we synchronize the large matrix and array between C++ DLL and VBA, and hence the result with dynamic length information is successfully transferred from C++ DLL to VBA.

With this Sync process, the Excel VRPTW application can be successfully built using the VBA call C++ DLL method.

4.3 PERFORMANCE OF THE EXCEL VRPTW APPLICATION

After bridging the Excel spreadsheet with VBA and linking VBA with C++ DLL, we can run the VRPTW application and the running time result can be obtained. The performance in terms of speed of the Excel VRPTW application will be evaluated by comparing with a C++ standalone application to solve all the 56 Solomon test cases. The Solomon test cases are well-established benchmark test cases for the VRPTW problem (Solomon, 1987). There are altogether 6 groups of test cases with different instances of vehicle capacity limit and Time Window limit. Each test case contains 100 customers. We expect to obtain the insights of the capability of this spreadsheet application through the comprehensive performance comparisons with the standalone application.

For each test case, the running times of 100 runs to solve specific test case are recorded to obtain the average running time of both application. Specifically, the Total time, reading, algorithm computing and writing time are compared to evaluate the performance. Here, we will present the performance comparison between the two applications on the 6 groups of test cases, as shown in the figure below. Within each group, the average running time of the group

are presented. The complete running time comparison for each test case can be found in APPENDIX D.

Table 4.1 Performance comparison of Excel VRPTW application and C++ standalone VRPTW application in Solomon test cases

Solomon Test Cases Group	Total time*		Reading		Algorithm computing		Writing	
	Excel	C++	Excel	C++	Excel	C++	Excel	C++
C101-C109	5.685	5.128	0.010	0.002	5.065	5.118	0.611	0.008
C201-C208	3.448	2.455	0.012	0.003	2.587	2.445	0.849	0.008
R101-R112	8.202	7.822	0.011	0.002	7.567	7.812	0.624	0.008
R201-R211	5.320	4.099	0.009	0.002	4.718	4.089	0.593	0.008
RC101-RC108	7.869	7.311	0.011	0.002	6.910	7.301	0.949	0.008
RC201-RC208	5.605	4.281	0.010	0.002	4.392	4.271	1.203	0.007

***:** *All the Units are in seconds*.

It can be seen that Excel VRPTW built with VBA call C++ DLL method and C++ VRPTW application show very similar performance on algorithm computing since the computations are both conducted in C++. However, the reading and writing time, especially the writing time of Excel VRPTW will be consistently longer than C++ VRPTW application. Therefore, this performance difference resulted in longer time in total for Excel VRPTW application.

Through the comprehensive comparison, the insights of the capability of spreadsheet applications built with VBA call C++ DLL method can be obtained: Excel VRPTW application is able to obtain good solution results at fast speed, which proves the capability of VBA call C++ DLL on building computational spreadsheet applications to solve complicated problems with sophisticated heuristics. Moreover, the performance difference compared with other standalone applications on text files will mainly come from the data transferring on Excel spreadsheet, especially, the writing on Excel.

4.4 CONCLUSIONS

In this Chapter, we built an Excel VRPTW application using VBA call C++ DLL method to solve the VRPTW problem, which is a much more complicated problem than TSP. Through the comprehensive comparison of the running time performance of this spreadsheet application with a C++ standalone VRPTW application on benchmark Solomon test cases, it is observed that the spreadsheet application built with VBA call C++ DLL method is capable to solve complicated problems with sophisticated heuristics with good performance. The powerful capability of the Hybrid method brings up the importance of building computational spreadsheet applications in research use.

Moreover, through the Excel VRPTW application example, we provide more specific guidelines of implementing VBA call C++ DLL method to build spreadsheet applications, and interfacing VBA with C++ DLL. The bridge between VBA and DLL is the most critical step of using this method to build applications on spreadsheet, and it is constructed from 3 aspects shown below:

(1) The exported function in C++ DLL matches the declared function in VBA;
(2) The function argument variables in C++ DLL match the function argument variables in VBA;
(3) The function return data type in C++ DLL match the return data type in VBA.

If the function is successfully matched and declared, we can transfer data between VBA and C++ DLL. Then we can use VBA to read data and write results on Excel spreadsheet, and use C++ DLL to compute the solution results.

Moreover, with the Sync process to overcome the inherent limitation of VBA call C++ DLL method that dynamic length array data are not able to transferred between VBA and C++ DLL, the route result with dynamic length information can be successfully transferred between C++ DLL and VBA. A computational spreadsheet application capable to solve complicated problems can be built successfully.

Chapter 5

Framework of Building Applications on Spreadsheet

5.1 INTRODUCTION

When building applications on spreadsheet, people have to make a choice among various options. Nowadays, when people select a specific option to build the spreadsheet application, they are primarily oriented by which method they are more familiar with, instead of choosing the most efficient and appropriate one that will satisfy their requirements. Therefore, how to select between different options becomes a very important and critical question needs to be investigated. Moreover, after selecting the method to build spreadsheet applications, for unsophisticated developers or inexperienced programmers, who are new to this method and have to start from the beginning to learn and apply this method to build spreadsheet applications, need efficient support and guidance for easier start.

Therefore, in this Chapter, based on the knowledge of the performance differences, ease of implementation, and the capabilities of different options investigated in Chapter 3 and Chapter 4, we construct a framework of building computational applications on spreadsheet to provide guidelines for people to select between various options under different scenarios. After a specific approach has been selected, we provide the structural routines and library codes of the specific method to support and guide people to build spreadsheet applications efficiently and conveniently.

The framework of building applications on spreadsheet consists of two parts. The first part provides the guidelines of how to select between the two options of spreadsheet platforms, Excel and Calc, under different scenarios. The second part provides the guidelines of how to select the most efficient method among different options to build applications on Excel or Calc under different scenarios.

We organize this Chapter as follows: in Section 5.2, we construct the framework of building applications on spreadsheet; section 5.3 illustrates the structures and routines of each method to provide people a much easier start to build spreadsheet applications with the options they have selected. In Section 5.4, summaries and conclusions are provided.

5.2 FRAMEWORK OF BUILDING APPLICATIONS ON SPREADSHEET

5.2.1 Selecting between Excel and Calc

Since saving of money and saving of time are usually the two major concerns of people to evaluate different options, we specify the cost and speed as two criteria to select between Excel and Calc spreadsheets.

To evaluate the cost of these two spreadsheet platforms, we compare the price of the latest version of these two spreadsheet software. To evaluate the speed of these two spreadsheets, we take the VBA performance on Excel spreadsheet and the OOO Basic performance on Calc spreadsheet in Chapter 3 to represent the speed performance of these two spreadsheets and make the comparison. As VBA and OOO Basic are both Internal programming methods officially integrated in Excel and Calc spreadsheets, their performances will be typical and reasonable to represent the speed performance of these two spreadsheets.

The table below shows the comparison of cost between Excel and Calc spreadsheets.

Table 5.1 Comparison of cost between Excel and Calc

	Microsoft Excel 2010	OpenOffice.org Calc 3.3.0
Cost	$139.99	Free

The Total running times of VBA on Excel and OOO Basic on Calc spreadsheet in various implementation tests with growing problem size and increasing algorithm complexity are shown in the table below.

Table 5.2 Comparison of speed performance between Excel and Calc

		Excel (VBA)	Calc (OOO Basic)
Sort	Small size (secs)	0.013	7.347
	Medium size	0.105	15.257
	Large size	0.550	96.611
Shortest Path	Small size	0.291	14.826
	Medium size	1.022	57.642
	Large size	8.159	*Out of Memory*
TSP	Small size	0.943	190.914
	Medium size	20.438	4791.056
	Large size	98.914	24433.080

It can be seen that with the increase of algorithm complexity and problem size, the speed performance of Excel is consistently faster than Calc spreadsheet. Moreover, Calc is not able to handle large number of elements in multi-dimensional matrices. With the growth of algorithm complexity, the speed performance of Calc becomes slower and slower and the gap to Excel spreadsheet becomes larger and larger. Therefore, as a spreadsheet platform to build computational applications, Excel has much faster speed than Calc spreadsheet.

In summary, although Calc is a free open-source spreadsheet software and Excel is not free to use, Excel's speed performance is much faster than Calc spreadsheet. Therefore, if people want to build applications on spreadsheet with free cost, Calc spreadsheet will be the right choice to satisfy their requirement. If there is no capital constraint and people want to build computational spreadsheet applications with fast speed performance, Excel will be the right option to choose to meet the requirement.

5.2.2 Selecting between Different Methods on Excel and Calc

Based on the spreadsheet platform selected, the next step will be to select the most effective method between different methods to build applications under different scenarios. In this section, based on the performance comparison of different methods on Excel and Calc spreadsheet in Chapter 3, we summarize the fast speed methods in different situations. The

different scenarios are the different combinations of two perspectives, the intensiveness of data transferring on spreadsheet, i.e., the intensiveness of reading and writing on spreadsheet, and the complexity of algorithm computing. When multiple options are available to be selected, based on the ease of implementation analysis of various methods, we select the most efficient method which requires the least implementation effort to build spreadsheet applications while has fast speed performance.

The performance comparison of different methods on Excel spreadsheet in Chapter 3 is summarized in Table 5.3 below.

Table 5.3 Performance comparison of different methods on Excel

		Methods on Excel			
		Internal	External		Hybrid
		VBA	VC++	Java	VBA call C++ DLL
Sort	Small size (secs)	0.0135	0.2693	0.1647	0.0064
	Medium size	0.105	2.6745	0.2813	0.0246
	Large size	0.5496	13.4682	1.1403	0.1039
Shortest Path	Small size	0.2911	0.6466	0.5437	0.4089
	Medium size	1.0219	1.5995	0.8240	0.9002
	Large size	8.1588	8.7816	3.2271	3.0023
TSP	Small size	0.9426	0.0720	0.2326	0.0449
	Medium size	20.4383	0.3900	1.7104	0.3258
	Large size	98.9141	1.6818	7.9047	1.5754

It can be seen that for applications with intensive data transferring but simple algorithm computing, such as Sort, VBA is the fast and good option to choose to build the application; for applications with less intensive data transferring but more complicated algorithm computing, such as Shortest Path, Java is the fast and good option to build the application; for applications with simple data transferring but complicated algorithm computing, such as TSP, VC++ is the fast and good option to build the application; and for applications with both

intensive data transferring and complicated algorithm computing, VBA call C++ DLL is the fastest and best option to build the application.

Hence, we can summarize the speed of different methods on Excel spreadsheet in the table shown below, where “Fast” indicates that the speed of this method is fast in this situation and empty space indicates that the method is slow in this situation.

Table 5.4 Speed of different methods to build applications on Excel under different criteria

	Methods on Excel			
	Internal	External		Hybrid
	VBA	VC++	Java	VBA call C++ DLL
Intensive Data Transferring	Fast			Fast
Complicated Algorithm Computing		Fast	Fast	Fast

The performance comparison of different methods on Calc spreadsheet in Chapter 3 is summarized in Table 5.5 below.

Table 5.5 Performance comparison of different methods on Calc

		Methods on Calc	
		Internal	External
		OOO Basic	Java
Sort	Small size (secs)	7.35	5.56
	Medium size	15.26	588.17
	Large size	96.61	13292.49
Shortest Path	Small size	14.83	16.16
	Medium size	57.64	72.23
	Large size	*Out of Memory*	2888.85
TSP	Small size	190.91	0.58
	Medium size	4791.06	6.65
	Large size	24433.08	27.90

It can be seen that for applications with intensive data transferring but simple algorithm computing, such as Sort, OOO Basic will provide much better performance. For applications with simple data transferring but complicated algorithm computing, such as TSP, Java will provide much better performance. For applications with less intensive data transferring and less complicated algorithm computing, such as Shortest Path, OOO Basic and Java will provide similar performance. However, as problem size increases, OOO Basic may become infeasible and Java will be the feasible option to build applications on Calc spreadsheet in this case.

Hence, we can summarize the speed performance of different methods on Calc spreadsheet in the table shown below, where "Fast" indicates that the speed of this method is fast in this situation and empty space indicates that the method is slow in this situation.

Table 5.6 Speed of different methods to build applications on Calc under different criteria

	Methods on Calc	
	Internal	External
	OOO Basic	Java
Intensive Data Transferring	Fast	
Complicated Algorithm Computing		Fast

When multiple options are available in a specific case, the ease of implementation, in terms of code structures and critical codes that need to be written, of different methods to build spreadsheet applications to unsophisticated programmers or inexperienced developers is summarized below.

Table 5.7 Ease of implementation of different methods to build spreadsheet applications

	Microsoft Excel				OpenOffice.org Calc	
	Internal	External		Hybrid	Internal	External
	VBA	VC++	Java	VBA call C++ DLL	OOO Basic	Java
Implementation Effort	Easy	Moderate	Moderate	Difficult	Easy	Moderate
Programming Skill requirement	Easy	Moderate	Moderate	Difficult	Easy	Moderate

As presented, Internal methods require the easiest effort to implement, External methods require a moderate level of effort to implement, and Hybrid methods require the most effort to implement to build applications on spreadsheet.

Thus, under scenarios of different combinations of two criteria, the intensiveness of data transferring and the complexity of algorithm computing, people are able to select the most efficient method that requires the least implementation effort to build spreadsheet applications while has fast speed performance.

5.2.3 The Framework

Thus far, we are able to propose a framework of building applications on spreadsheet that provides guidelines for people to select between different options of spreadsheet platforms and methods. To be specific, the framework will guide people to select between Excel and Calc spreadsheets, and select the most efficient method within different options in different scenarios, as shown in Figure 5.1 below.

It can be seen that firstly, to select between Excel and Calc to build the application, if fast speed is required, Excel will be the right choice, otherwise if free software is needed, Calc will be the right selection.

Secondly, to select between different methods on each spreadsheet, for Excel, if the problem is very easy, VBA will be the most efficient method. Otherwise if the problem is not very easy and there will be intensive reading and writing on spreadsheet, VBA call C++ DLL is the best choice. If the problem is complicated but data transferring is easy, C++ will be the most efficient option. Otherwise if there is no intensive data transferring and the problem is not very complicated, Java will be the right choice in this case. For Calc, Java will be the most efficient option when the problem is complicated and there is no intensive data transferring. OOO Basic will be the most efficient method when the problem is not complicated.

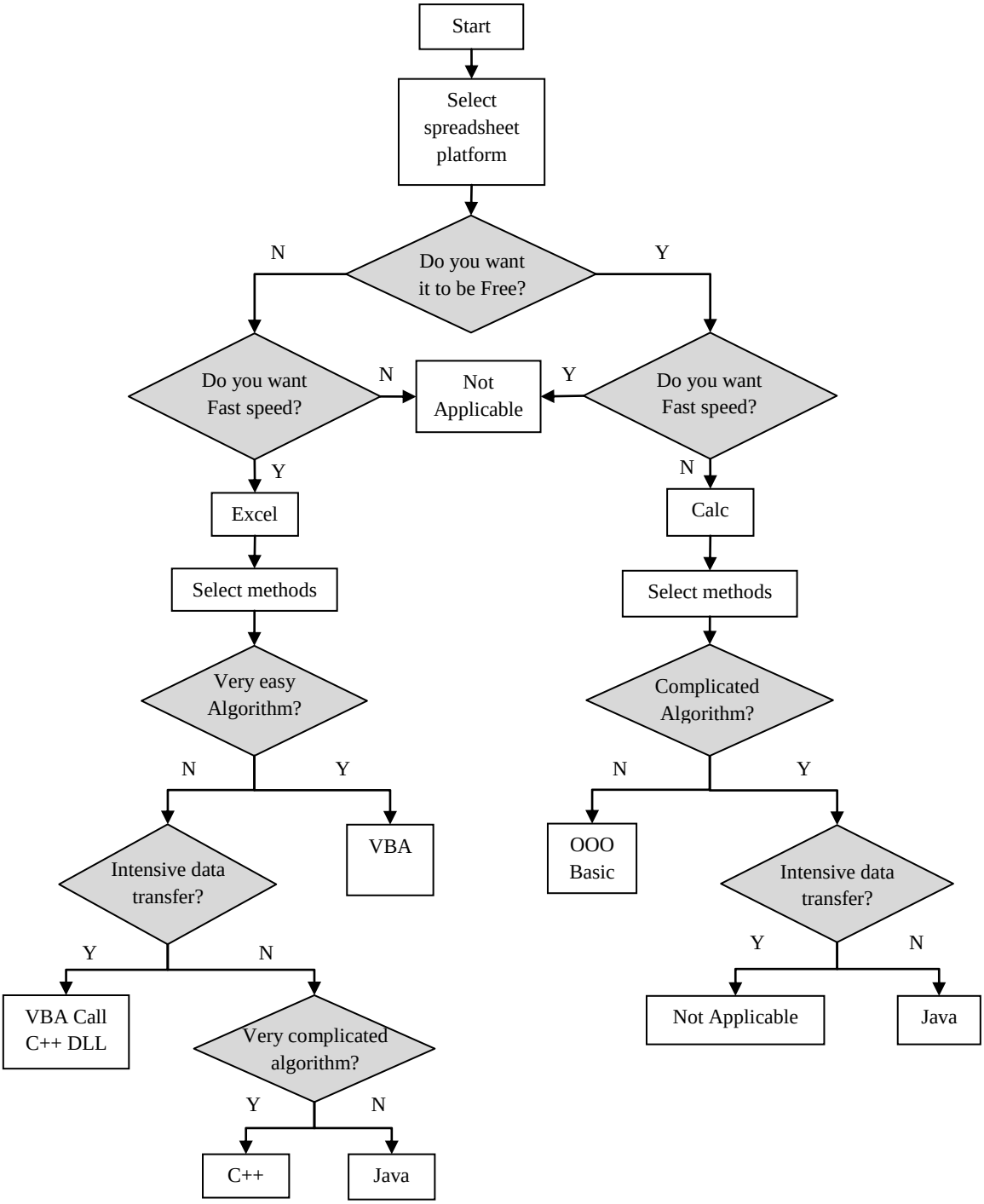

Figure 5.1 Framework of building applications on spreadsheet

With this framework, people are able to make right and appropriate decisions under different scenarios from the start to the end of the process of building applications on spreadsheet, which greatly improves the efficiency of building spreadsheet applications.

5.3 STRUCTURES AND ROUTINES OF DIFFERENT METHODS

With the selection of a specific method to build applications on the chosen spreadsheet platform, for unsophisticated developers or inexperienced programmers, they may have to put in a lot of effort and spend a lot of time to become familiar with this method to build the spreadsheet applications, which is very inefficient and inconvenient.

To address this inefficiency, we provide in this section the structures and routines for each method, and the library codes of comprehensive implementation examples to help people to obtain a much easier start to build spreadsheet applications.

To build applications on spreadsheet with the methods discussed above, we have to in general build up the structures of codes as shown in Figure 5.2 shown below.

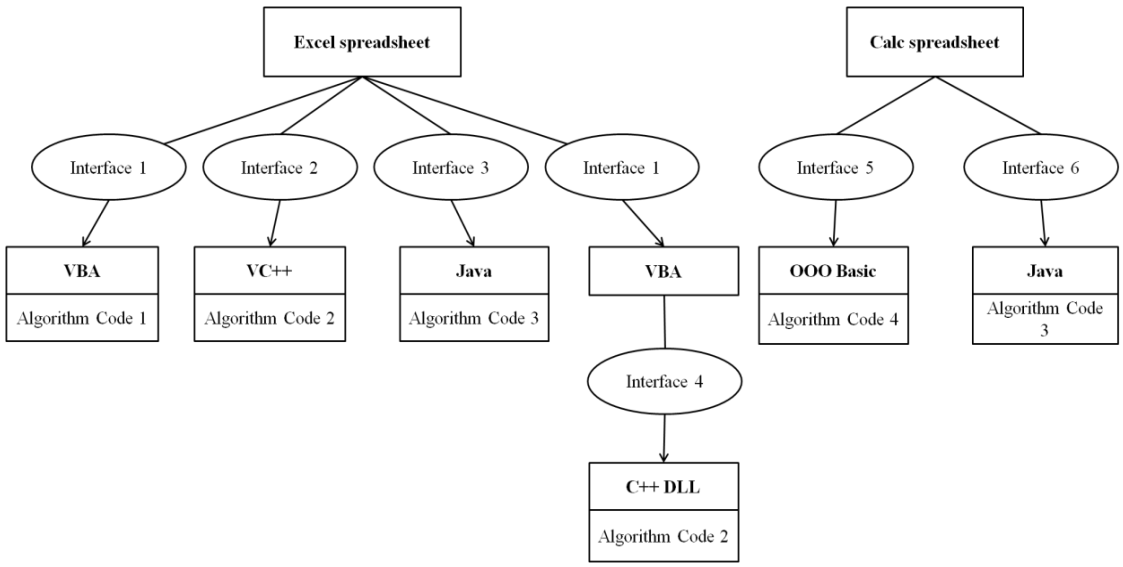

Figure 5.2 Code Structures of different methods to build applications on spreadsheet

It can be seen that for different methods on different spreadsheets, we have to build up different interfaces to bridge these methods with spreadsheet software. With the bridging interfaces, we are able to transfer input data from spreadsheet to programming methods. Then we have to construct the codes of the Algorithm to carry out algorithm computations and obtain the results. Finally, the results are written back to spreadsheet through the bridging interface. Moreover, for the Hybrid programming method, VBA call C++ DLL is able to use

the same interface as VBA and the same codes of the algorithm as VC++, with the difference that we have to build up an additional bridging interface to link VBA with C++ DLL to transfer data between them.

To build up the interface between different methods and spreadsheets, we have to in general obtain the three layers of access to the spreadsheet step by step, as shown below:

Step 1. Obtain the access of *SpreadsheetDocument* or *Workbook*;

Step 2. Obtain the access of *Sheets* or *Worksheets*;

Step 3. Obtain the access of *Cells*.

To build up the interface of the Hybrid method between the Internal and External methods, we have to in general make sure that these two methods are matched in three aspects:

(1) The exported function in C++ DLL matches the declared function in VBA;
(2) The function argument variables in C++ DLL matches the function argument variables in VBA;
(3) The function return type in C++ DLL matches the return type in VBA.

To construct these interfaces, different methods provide and support different components and Class functions to achieve this goal. The elements, Classes or functions that support accessing three levels of spreadsheet in each method were introduced in Chapter 3 with sample codes. Moreover, in this research, comprehensive implementations including Sort, Shortest Path, TSP, and VRPTW, are constructed to provide sufficient library codes as references. Detailed routines to build spreadsheet applications with different methods and the library codes of Sort, Shortest Path, TSP and VRPTW implementations are illustrated in APPENDICES A, B and C.

5.4 SUMMARIES AND CONCLUSIONS

In this Chapter, based on performance differences and ease of implementation of different options, the framework of building applications on spreadsheet is constructed to provide

guidelines to select between different options of spreadsheet platforms and methods to build applications on spreadsheet. The structures, routines of different methods to build spreadsheet applications and the library codes of comprehensive implementation examples are provided for people to build applications on spreadsheet much more efficiently and conveniently. It will greatly reduce and save the time to implement a new method for building spreadsheet applications, especially for unsophisticated developers and inexperienced programmers. This framework supports the decision making from the start to the end of the building process and it helps people to select the most efficient options under different scenarios. Hence, people can build the spreadsheet applications in a highly efficient way.

Chapter 6

Conclusions and Future Research

6.1 INTRODUCTION

This study was motivated by the observation from the literature that there are deficiencies in knowledge to select between options that people encountered when building applications on spreadsheet under different scenarios. The objective of the study is to investigate the performance of different methods on spreadsheet and construct a general framework of building applications on spreadsheet to provide guidelines for people to build applications more efficiently and conveniently. To this aim, we have formulated five research questions in Chapter 1. To answer these questions, we have implemented comprehensive tests to compare the performance of different methods on spreadsheet and important properties of spreadsheet applications. This study has provided useful guidance from the start to the end of the building application process with routines and library codes. In this chapter, we summarize our major contributions and significances of this study (section 6.2), followed by the discussion of the limitations of this study and directions for future research (section 6.3).

6.2 MAJOR CONTRIBUTIONS

This study has investigated several topics on building applications on spreadsheet. Specifically, the major contributions are described below,

Firstly, a comprehensive comparison of different kinds of methods, including Internal programming methods, External programming methods and Hybrid programming methods to build OR/MS applications on spreadsheet is conducted. The performance of VBA, VC++, Java, VBA call C++ DLL on Excel and OOO Basic, Java on Calc spreadsheet are compared in terms of running time results of comprehensive implementation tests with growing problem size and increasing algorithm complexity. Through the performance comparison on these methods,

their performance differences are revealed and their strengths and weaknesses are shown. The results of the comparative study in Chapter 3 provide useful information to select between different options to build spreadsheet applications. The findings in this study suggest that Hybrid programming methods which combine the efficiency of data transferring and algorithm computing, such as VBA call C++ DLL on Excel, will give the fastest speed for spreadsheet applications. Besides, the ease of implementation of different methods in terms of the code structures and the critical codes needed to build spreadsheet applications is analyzed. Hence people can select the option that achieves the performance requirement while requires the least implementation effort.

Secondly, a spreadsheet application example of solving the complicated VRPTW problem is built to illustrate the capability of the VBA call C++ DLL method on Excel spreadsheet. The VRP problem with Time Windows solved by the tabu-search heuristic has modeled various complexities in reality. The success of the Excel VRPTW application built with VBA call C++ DLL method shows the capability of this method to build spreadsheet applications to solve very complicated problems with sophisticated algorithms in a short period of time. Furthermore, the bridging interface between VBA and C++ DLL is illustrated in detail to provide useful guidelines for people to apply this method to build spreadsheet applications. Moreover, we propose a Sync process which overcomes the inherent limitation of VBA call C++ DLL method for being unable to transfer data arrays with dynamic length.

Thirdly, a framework of building computational applications on spreadsheet was constructed to provide guidelines of how to select the most efficient option among various options in different scenarios. Based on the performance differences and ease of implementation of different methods on spreadsheet in Chapter 3 and the insight of the capability of VBA call C++ DLL method to build spreadsheet applications, we identify different criteria and scenarios and compare different options to provide guidelines of selecting the most efficient method to build spreadsheet applications. With this framework, people are able to build spreadsheet applications with high efficiency.

Lastly, the structures, routines of different methods and library codes of comprehensive implementation examples are provided for people to build applications with the method and spreadsheet platform selected easily and conveniently.

6.3 LIMITATIONS AND FUTURE RESEARCH

This study focuses on building applications on spreadsheet. Based on the spreadsheet software, applications are widely built on spreadsheet to conduct data analysis and further extend the functional capability of spreadsheet due to the advantages of spreadsheet on ease of use and user friendliness. However, spreadsheet applications also have various kinds of limitations as illustrated below:

Firstly, Spreadsheet applications have the difficulty and limitation with Real-time data (RTD). Real-time data is data that updates immediately after collection, such as Stock Quotes, Web server Loads, etc. Hence, such kind of data will change and update frequently, and the application built on spreadsheet needs to perform calculations and obtain results without much delay. In this case, the spreadsheet applications will have some common limitations for real-time solutions, such as, updates are missed easily, and the solutions are inefficient, which will cause a lot of difficulties for computations based on real-time data.

Secondly, Spreadsheet applications are error-prone. In programming, we will follow strict discipline to prevent most errors caused by humans. However, the customized developing process of applications on spreadsheet is informal and not always well-structured. Hence, spreadsheet applications will be error-prone, especially when data size is very large. The results of spreadsheet applications will be invalid if the calculation is based on wrong numbers.

In this research, a framework is proposed to provide useful guidelines for the selection of methods to build applications on spreadsheet under different scenarios by conducting empirical studies with implementations on different spreadsheet platforms, using different methods, and solving problems of different complexity with selected algorithms. However, there are still

several limitations of this study need to be mentioned and some interesting areas deserve further explorations, as illustrated below:

Firstly, The selected problems are mainly in the area of operations research and management science. It will be much more comprehensive if more application examples in various areas can be provided such that the findings of this study, such as the framework, can be more generalized.

Secondly, Hybrid programming methods on Calc spreadsheet is not investigated in this study. Hence, finding possible solutions to build applications on Calc spreadsheet using Hybrid programming methods will be a very interesting research topic as it will make the framework complete. If Hybrid methods, which combine the advantages of Internal and External methods can be found, the performance of applications on Calc spreadsheet can be greatly improved.

Thirdly, in this research, VRPTW, as a complicated problem, is used to test the capability of VBA call C++ DLL method to build spreadsheet applications. However, implementation examples to solve difficult problems that are even more complicated than VRPTW can further extend the knowledge of the capability of VBA call C++ DLL methods to build spreadsheet applications.

Lastly, Other than Reading and Writing data on spreadsheet directly, the data in spreadsheet can be transferred indirectly, such as using a text file as an intermediate means to transfer data. Thus, we can first use VBA and OOO Basic, which are integrated in spreadsheet, to read data from the spreadsheet to the text file. Then, we can adopt VC++ or Java to read data from the text file, carry out the algorithm computation, and write the results back to text files. After that, we can again use VBA and OOO Basic to transfer the results in the text file to the spreadsheet and complete the whole process. Similar as VBA call C++ DLL introduced in this research framework, this method also combines advantages of VBA and OOO Basic on data transferring on spreadsheet and VC++ and Java on algorithm computing. Implementing this method of building spreadsheet applications is a useful topic that deserves attention.

REFERENCES

Anthony, S., Wilson, J. (1990). Manpower modelling using a spreadsheet. Omega Int. J. of Mgmt Sci. Vol. 18, No. 5 , 505-510.

Archer, N. P. (1989). Electronic Spreadsheet Structures. Computers & Operations Research, Vol. 16, Issue 5 , 493-496.

Au, S.K., Cao, Y., Wang, Z.J. (2010). Implementing advanced Monte Carlo simulation under spreadsheet environment. Structural Safety, Vol. 32, 281–292.

Billo, E. J. (2001). Excel for Chemists-A Comprehensive Guide (2nd Edition). New York: John Wiley & Sons.

Bloch, S. (1995). Spreadsheet Analysis For Engineers And Scientists. New York: John Wiley & Sons.

Chehab, A., El-Hajj, A., Al-Husseini, M., & Artail, H. (2004). Spreadsheet Applications in Electrical Engineering: A Review. Int. J. Engng Ed. Vol. 20, No. 6 , 902-908.

Composition Tool. IEEE Transactions on Service Computing, vol. 1, No. 4.

Conway, D., Ragsdale, C.T. (1997). Modeling optimization problems in the unstructured world of spreadsheets. Omega, Int. J. Mgmt Sci. Vol. 25, No. 3 , 313-322.

David ,N., Ragsdale, C.T. (2003). A decision support methodology for stochastic multi-criteria linear programming using spreadsheets. Decision Support Systems, Vol.36, 99– 116.

Dianond, D., Hanratty, V. C. (1997). Spreadsheet Applications in Chemistry using Microsoft Excel. New York: John Wiley & Sons.

Earnest, D. L. (1987). Capital Equipment Justification: A Spreadsheet Application Template. Computers & Industrial Engineering, Vol. 13, Nos 1-4 , 341-345.

Fields, F. (1986). Industrial Engineering Use of A Spreadsheet. Computers & Industrial Engineering, Vol. 11, Issues 1-4 , 312-315.

Filby, G. (1998). Spreadsheets in Science and Engineering. Berlin Heidelberg: Springer.

Gansel, B. B. (2008). About the Limitations of Spreadsheet Applications in Business Venturing. in Operations Research Proceedings, ser. Operations Research Proceedings, J. Kalcsics and S. Nickel, Eds., vol. 2007. Berlin, Heidelberg: Springer, pp. 219–223.

Hazel, A. L. (n.d.). A brief introduction to C++ and Interfacing with Excel. Retrieved from http://www.maths.manchester.ac.uk/~ahazel/EXCEL_C++.pdf

Hesse, R., Scerno, D. H. (2009). How Electronic Spreadsheets Changed the World. Interfaces, Vol. 39, No. 2 , 159-167.

Ionut Arghire. (2012) "Microsoft's Office Has over One Billion Users". SOFTPEDIA News. World Wide Web, http://news.softpedia.com/news/Microsoft-s-Office-Has-Over-One-Billion-Users-280426.shtml.

Jr, J. S. (1987). Industrial engineering spreadsheet applications from a manufacturing resource planning (MRP-II) system. Computers & Industrial Engineering, Vol. 13, Issues 1-4 , 100-106.

Kharab, A. (2000). An advanced macro spreadsheet program for the simplex method. Computers & Operations Research, Vol. 27 , 233-243.

Kokol, P. (1989). Application of Spreadsheet Software in Software Engineering Measurement Technology. Information and Software Technology, Vol. 31, Issue 9 , 477-485.

Lau, H.C., M. Sim, K.M. Teo. (2003). Vehicle routing problem with time windows and a limited number of vehicles. European Journal of Operational Research, vol. 148, No. 3, 559–569,

LeBlanc, L. J., and Galbreth, M. R. (2007). Implementing Large-Scale Optimization Models in Excel Using VBA. Interfaces, Vol. 37 , 370-382.

Lynne,S., John, P. (2004). Using spreadsheet solvers in sample design. Computational Statistics & Data Analysis, Vol. 44, 527 – 546.

Obrenovic, Z., Gasevic, D. (2008) . End-User Service Composition: Spreadsheets as a Service

Oke, S. (2004). Spreadsheet Applications in Engineering Education: A Review. Int. J. Engng Ed. Vol. 20, No. 6 , 893-901.

Parlar, M. (1986). Dynamic Programming On an Electronic Spreadsheet. Computers & Industrial Engineering, Vol. 10, No. 3 , 203-213.

Power, D. J. (2004). "A Brief History of Spreadsheets". DSSResources.com, World Wide Web, http://dssresources.com/history/sshistory.html, version 3.6.

Punuru, J. R., Knopf, F. C. (2008). Bridging Excel and C/C++ Code. Comput Appl Eng Educ, Vol. 16 , 289-304.

Raffensperger, J. F. (2003). New Guidelines for Spreadsheets. International Journal of Business and Economics, Vol. 2, No. 2 , 141-154.

Ragsdale, C. T. (2001). Spreadsheet Modeling and Decision Analysis (3rd Edition). Ohio: South-Western Publishing.

Rosen, E. (2001). Use of C++ Dlls in Visual Basic for Applications With Excel 2000. Retrieved from CACHE: http://cache.org/site/news_stand/spring01/spring2001_useofc.pdf

Rosen, E., Adams, R. (1987). A Review of Spreadsheet Usage In Chemical Engineering Calculations. Computers & Chemical Engineering, Vol. 11, No. 6 , 723-736.

Rosen, E., Partin, L. (2000). A Perspective: The Use of the Spreadsheet for Chemical Engineering Computations. Ind. Eng. Chem. Res., Vol. 39, No. 6 , 1612-1613.

Roy, A., Lasdon, L., Plane, D. (1989). End-user optimization with spreadsheet models. European Journal of Operational Research, Vol. 39, Issue 2 , 131-137.

Sakalli U. S. and Birgoren B., (2009). A spreadsheet-based decision support tool for blending problems in brass casting industry, Computers & Industrial Engineering, 56, 724–735.

Solomon M. M. (1987). Algorithms for the vehicle Routing and Scheduling Problem with Time Window Constraints, Operations Research, 41, 469-488.

Stan J. Liebowitz, Stephen Margolis. (2001). Winners, losers & Microsoft: competition and antitrust in high technology. Independent Institute. Chapter 8, 178-193.

Thomas, A. Grossman & Ozgur Ozluk (2010). Spreadsheets Grow Up: Three Spreadsheet Engineering Methodologies for Large Financial Planning Models. Proc. European Spreadsheet Risks Int. Grp. (EuSpRIG), 1-15.

Walkenbach, J. (2004). Excel 2003 Power Programming With VBA. New York: Wiley.

Whitehouse, G., & Hodak, G. (1986). A Spreadsheet-Based Macro Manpower Model . Computers and Industrial Engineering, Vol. 11, Issue 1-4 .

Yap, C.W. (2006). BasicExcel - A Class to Read and Write to Microsoft Excel. Retrieved from The Code Project http://www.codeproject.com/Articles/13852/BasicExcel-A-Class-to-Read-and-Write-to-Microsoft

Zimmerman, S. M., & Gibson, D. R. (1989). A Proposed Method to Use Electronic Spreadsheets to Develop Quality Control Charts. Computers & Industrial Engineering, Vol. 17, No. 1-4 , 384-389.

APPENDICES

APPENDIX A

Routines of Different Methods to Build Applications on Excel Spreadsheet

A.1 INTERNAL PROGRAMMING METHOD ON EXCEL

A.1.1 VBA Routine

(1) Generate an Input file in Microsoft Excel, Save as Excel Macro-Enabled Workbook, such as "Input.xlsm" file.

(2) Open Visual Basic Editor.

In Excel 2003 and earlier, click **Tools → Macro → Visual Basic Editor** menu item to get into the VBE.

In Excel 2007, Click **Office → Excel Options → Popular tab →** Select **"Show Developer tab in the Ribbon"** will make the Developer ribbon visible. Then click **Developer Ribbon → Visual Basic** button to get into the VBE.

For a quicker access, Press **Alt+F11**.

(3) Insert a Module

To start building the application, we usually need at least one module in a project where one will typically store one's codes. To insert a module, click the **Insert → Module** menu item.

(4) Write the codes of Application

a) Construct the Interface between VBA and Excel Spreadsheet (Reading & Writing)

To build up the Interface between VBA and Excel Spreadsheet, we need to obtain three levels of access on Excel Spreadsheet step by step.

Step 1. Obtain the Access of Workbook layer In VBA, we can use Application's property and function and declare (Dim) a Workbook type variable to obtain the access of Workbook level.

Step 2. Obtain the Access of Worksheet layer. After Workbook, we can use Workbook's property and function and declare (Dim) a Worksheet type variable to obtain the access of Worksheet level.

Step 3. Obtain the Access of Cell level. After Worksheet, we can access the Cells by using Worksheet variable's Cell function. Usually, we can use "Cells(i, j).Value" to read a single value stored in Cell(i, j), where i denotes the row index and j denotes the column index. In this way, the input data are read cell by cell, or we can use "Range(Cells(i_1, j_1), Cells(i_2, j_2)).Value" to read a group of data stored between Cells(i_1, j_1) and Cells(i_2, j_2). In this way, the input data are read group by group.

The Interface sample codes between VBA and Excel Spreadsheet in Sort, Shortest Path, and TSP application examples are shown in the figures below.

```
Dim work_book As Workbook
Dim sheet As Worksheet
Dim data As Variant

'/* Access the Workbook level */
Set work_book = ActiveWorkbook
'/* Access the Sheets level */
Set sheet = work_book.Worksheets("Input")
'/* Access the Cells level */
data = sheet.Range(Cells(2, 1), Cells(n + 1, 1)).Value
For i = 1 To datasize
   arr(i - 1) = data(i, 1) '/* or arr(i - 1) = Cells(i, 1).Value */
Next i
```

Figure A.1 Interface between VBA and Excel in Sort implementation: Read data

```
Dim result As Variant

'/* Access the Workbook level */
Set work_book = ActiveWorkbook
'/* Access the Sheets level */
Set sheet = work_book.Worksheets("Sheet1")
'/* Access the Cells level */
result = sheet.Range(Cells(2, 2), Cells(n + 1, 2)).Value
For i = 1 To datasize
   result(i, 1) = arr(i - 1) '/* or  Cells(i, 2).Value =arr(i - 1) */
Next i
sheet.Range(Cells(2, 2), Cells(n + 1, 2)).Value = result
```

Figure A.2 Interface between VBA and Excel in Sort implementation: Write result

```
Dim work_book As Workbook
Dim sheet As Worksheet

'/* Access the Workbook level */
Set work_book = ActiveWorkbook
'/* Access the Sheets level */
Set sheet = work_book.Worksheets("Input")
maxRows = sheet.UsedRange.Rows.count
For i = 1 To maxRows - 1
   '/* Access the Cells level */
   p = Cells(i + 1, 1).Value
   q = Cells(i + 1, 2).Value
   t = Cells(i + 1, 3).Value
   graph(p, q) = t
   graph(q, p) = t
Next i
```

Figure A.3 Interface between VBA and Excel in Shortest Path implementation: Read data

```
'/* Access the Workbook level */
Set work_book = ActiveWorkbook
'/* Access the Sheets level */
Set sheet = work_book.Worksheets("Output")
For j = 1 To n
   sheet.Cells(j + 1, 2).Value = weight(j - 1)
Next j
For j = 1 To n
   sheet.Cells(j + 1, 1).Value = j - 1
Next j
For i = 0 To n - 1
   For j = 0 To count(i) - 1
        sheet.Cells(i + 2, j + 3) = getpath(i, j)
   Next j
Next i
```

Figure A.4 Interface between VBA and Excel Spreadsheet in Shortest Path implementation: Write result

```
Dim work_book As Workbook
Dim sheet As Worksheet

'/* Access the Workbook level */
Set work_book = ActiveWorkbook
'/* Access the Sheets level */
Set sheet = work_book.Worksheets("Input")
For i = 1 To n
   '/* Access the Cells level */
   point(i - 1, 0) = sheet.Cells(i + 1, 1).Value
   point(i - 1, 1) = sheet.Cells(i + 1, 2).Value
Next i
```

Figure A.5 Interface between VBA and Excel Spreadsheet in TSP implementation: Read data

```
'/* Access the Workbook level */
Set work_book = ActiveWorkbook
'/* Access the Sheets level */
Set sheet = work_book.Worksheets("Input")
'/* Access the Cells level */
sheet.Cells(2, 1).Value = TotalDistance
For i = 1 To n
   sheet.Cells(i + 1, 2).Value = Route(i - 1)
Next i
sheet.Cells(n + 1, 2).Value = 0
```

Figure A.6 Interface between VBA and Excel Spreadsheet in TSP implementation: Write result

b) Construct the Algorithm Computing codes

Through the Interface between VBA and Excel Spreadsheet, the Input data can be read, and we can next construct the algorithm's structure to compute and obtain the result.

(5) Run the application in Excel Spreadsheet

After finishing all the above steps, we can run the application in Excel Spreadsheet to complete building applications on Excel Spreadsheet with VBA method.

A.2 EXTERNAL PROGRAMMING METHOD ON EXCEL

A.2.1 VC++ Routine

(1) Generate an Input file in Microsoft Excel, Save as Excel 97-2003 Workbook, such as "Input.xls" file.

(2) Under Microsoft Visual Studio C++ 2010, **create a new C++ project**.

Click the menu item **File → New → Project →** Select **Win32 Console Application → Enter Project name →** Click **OK → Next →** Select **Console Application** & **Empty Project → Finish**.

(3) Add Open-Source Library into the C++ project

In VC++ Routine on Excel, the Interface between C++ and Excel Spreadsheet such as reading and writing on Excel are fulfilled using an open-source library called **BasicExcel** downloaded from the Internet.

Click the menu item **Project → Add Existing Item** (Shift+Alt+A) **→** Select **BasicExcel.hpp & BasicExcel.cpp** file **→ Add**.

(4) Create a new C++ (.cpp) file

The ".cpp" file created will be the place to build smart applications, and the number of ".cpp" files needed will depend on the application's requirement.

Click the **Project → Add New Item** menu item (Ctrl+Alt+A) **→** Select **C++ File (.cpp) → Enter file name → Add**.

(5) Write the Codes of Application

a) Including file and namespace

We will use the Classes and Functions defined in BasicExcel.hpp and BasicExcel.cpp files to build up the Interface between C++ and Excel Spreadsheet. Hence, we need to include "BasicExcel.hpp" first, while the other include files needed will depend on the application's requirement. This is shown in the Figure below.

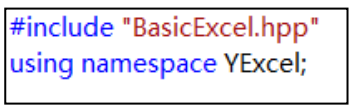

Figure A.7 C++ on Excel Spreadsheet: Include files and namespace

b) Construct the Interface between C++ and Excel Spreadsheet (Reading & Writing)

To build up the Interface with reading and writing on Excel Spreadsheet, we need to obtain three levels of access on Excel step by step.

Step 1. Load Workbook file. The access of Workbook can be obtained by calling the *"Load()"* function in Class "*BasicExcel*".

Step 2. Access Worksheet. The access of Worksheet can be obtained by calling the "*GetWorksheet()*" function in Class "*BasicExcel*" and passing to a pointer declared as "*BasicExcelWorksheet*" object.

Step 3. Access Cells. The access of Cells can obtained by calling the function "*Cell(i, j)*" in Class "*BasicExcelWorksheet*" and received by declaring a pointer to "*BasicExcelCell*" object, where *i* stands for row index and *j* stands for column index in *Cell(i, j)* starting from 0. After obtaining the access of Cells, the value stored in Cells can be retrieved by calling functions such as "*GetInteger()*", "*GetDouble()*", or "*GetString()*".

The Interface sample codes between C++ and Excel Spreadsheet in Sort, Shortest Path, and TSP application examples are shown in the figures below.

```
// Interface between C++ and Excel Spreadsheet: Read data from Excel //
BasicExcel e;
// Obtain the Access of Excel Workbook //
e.Load("D:\\Sort.xls");
// Obtain the Access of Worksheet //
BasicExcelWorksheet* sheet1 = e.GetWorksheet("Sheet1");
if (sheet1)
{
   for (size_t i=0; i<n; ++i)
   {
      // Obtain the Access of Cells //
      BasicExcelCell* cell = sheet1->Cell(i+1,0);
      arr[i]=cell->GetInteger();
   }
}
```

Figure A.8 Interface between C++ and Excel Spreadsheet in Sort implementation: Read data

```
// Interface between C++ and Excel Spreadsheet: write result to Excel //
BasicExcel e;
// Obtain the Access of Excel Workbook //
e.Load("D:\\Sort.xls");
// Obtain the Access of Worksheet //
BasicExcelWorksheet* sheet1 = e.GetWorksheet("Sheet1");
for (size_t i=0; i<n; ++r)
{
   // Obtain the Access of Cells //
   BasicExcelCell* cell = sheet1->Cell(i+1,1);
   cell->SetInteger(arr[i]);
}
// Save the Changes //
e.Save();
```

Figure A.9 Interface between C++ and Excel Spreadsheet in Sort implementation: Write result

```
// Interface between C++ and Excel Spreadsheet: Read data  //
BasicExcel e;
// Obtain the Access of Excel Workbook //
e.Load("D:\\ShortestPath.xls");
// Obtain the Access of Worksheet //
BasicExcelWorksheet* sheet1 = e.GetWorksheet("Input");
if (sheet1)
{
   size_t maxRows = sheet1->GetTotalRows();
   for (size_t i=0; i<maxRows; ++i)
   {
      int p,q;
      double t;
      // Obtain the Access of Cells //
      BasicExcelCell* cell1 = sheet1->Cell(i+1,0);
      BasicExcelCell* cell2 = sheet1->Cell(i+1,1);
      BasicExcelCell* cell3 = sheet1->Cell(i+1,2);
      p=cell1->GetInteger();
      q=cell2->GetInteger();
      t=cell3->GetDouble();
      graph[p][q]=t;
      graph[q][p]=t;
   }
```

Figure A.10 Interface between C++ and Excel Spreadsheet in Shortest Path implementation: Read data

```
// Interface between C++ and Excel Spreadsheet: Write result to Excel  //
BasicExcel e;
// Obtain the Access of Excel Workbook //
e.Load("D:\\ShortestPath.xls");
// Obtain the Access of Worksheet //
BasicExcelWorksheet* sheet2 = e.GetWorksheet("Output");
for (size_t i=0; i<n; ++i)
{
   BasicExcelCell* cell1 = sheet2->Cell(i+1,0);
   cell1->SetInteger(i);
   BasicExcelCell* cell2 = sheet2->Cell(i+1,1);
   cell2->SetDouble(weight[i]);
   for (size_t j=0; j<c[i]; j++)
   {
      BasicExcelCell* cell3 = sheet2->Cell(i+1,j+2);
      cell3->SetInteger(getpath[i][j]);
   }
}
e.SaveAs("D:\\ShortestPath.xls");
```

Figure A.11 Interface between C++ and Excel Spreadsheet in Shortest Path implementation: Write result

```
// Interface between C++ and Excel Spreadsheet: Read data from Excel  //
BasicExcel e;
// Obtain the Access of Workbook //
e.Load("D:\\TSP.xls");
// Obtain the Access of Worksheet //
BasicExcelWorksheet* sheet1 = e.GetWorksheet("Input");
if (sheet1)
{
   for (size_t i=0; i<n; ++i)
   {
      // Obtain the Access of Cells //
      BasicExcelCell* cell1=sheet1->Cell(i+1,0);
      BasicExcelCell* cell2=sheet1->Cell(i+1,1);
      pointx[i]=cell1->GetDouble();
      pointy[i]=cell2->GetDouble();
   }
}
```

Figure A.12 Interface between C++ and Excel Spreadsheet in TSP implementation: Read data

```
// Interface between C++ and Excel Spreadsheet: Write result to Excel //
BasicExcel e;
// Obtain the Access of Workbook //
e.Load("D:\\TSP.xls");
// Obtain the Access of Worksheet //
BasicExcelWorksheet* sheet2 = e.GetWorksheet("Output");
if(sheet2)
{
   BasicExcelCell* cell1=sheet2->Cell(1,0);
   cell1->SetDouble(TotalDistance);
   for (size_t i=0; i<n; ++i)
   {
      BasicExcelCell* cell2=sheet2->Cell(i+1,1);
      cell2->SetInteger(Route[i]);
   }
   BasicExcelCell* cell3=sheet2->Cell(n,1);
   cell3->SetInteger(0);
}
```

Figure A.13 Interface between C++ and Excel Spreadsheet in TSP implementation: Write result

c) Construct Algorithm Computing Engine

Through the Interface between C++ and Excel Spreadsheet, the Input data can be read, and we can next construct the algorithm's structure to compute and obtain the result.

(6) Build and Generate the C++ EXE file

Click the menu item **Build → Build Solution** (F7), and the C++ EXE file will be generated in the Folder of C++ project.

(7) Run the application in Excel Spreadsheet

In order to run the C++ EXE file from Excel Spreadsheet, we will create a macro to reference the C++ EXE file and run the Macro to trigger the C++ EXE program. We will write the codes as shown in the Figure below.

```
Sub CPlusSortMacro()

Dim retVal As Variant
retVal = Shell("D:\Excel\C++\C++_SORT\Sort.exe", vbNormalFocus)

End Sub
```

Figure A.14 Excel Macro to trigger C++ run in Sort implementation

Hence, the flow of steps is Create an Excel Macro-Enabled Workbook → Enter the **VBE** from Developer Ribbon → Insert a **Module** to create a Macro → Reference the C++ EXE file inside the Macro → Run the Macro to trigger the C++ EXE program to run.

A.2.2 Java Routine

(1) Generate an Input file in Microsoft Excel, Save as Excel 97-2003 Workbook, such as "Input.xls" file.

(2) Under Eclipse IDE for Java Developers, **Create a new Java project**

Click the menu item **File → New → Java Project → Enter Project name → Next → Finish**.

(3) Add open-source library into the Java project

In Java Routine on Excel spreadsheet, the Interface between Java and Excel Spreadsheet such as reading and writing on Excel is fulfilled by using an open-source library called **JXL** downloaded from the Internet.

Click the menu item **Project → Properties → Java Build Path →** Select the label **Libraries →** Click **Add External JARs →** Select **"jxl.jar" → OK**.

(4) Create a new Class (.java) file

The Class file created will be the place to develop smart applications, and the number of Class files needed will depend on the application's requirement.

Right Click the Java Project created → **New** → **Class** → **Enter Class name** → Select **"public static void main(String[] args)"** (Optional) → **Finish**.

(5) Write the Codes of Application

a) Import Open-Source Library Classes

We will use the Classes and Functions defined in jxl.jar file to build up the Interface between Java and Excel Spreadsheet. Hence, we have to import the JXL Library first. The other import Libraries needed will depend on the application's requirement. This is shown in the Figure below.

```
import jxl.*;
import jxl.write.*;
import java.io.*;
```

Figure A.15 Java on Excel Spreadsheet: Import Classes

b) Construct the Interface between Java and Excel Spreadsheet (Reading & Writing)

To build up the Interface with reading and writing on Excel Spreadsheet, we need to obtain three levels of access on Excel step by step.

Step 1. Access the Workbook. The access of Workbook can be obtained by calling the *"getWorkbook()"* function and passing to the Class *Workbook* object.

Step 2. Access the Worksheet. The access of Worksheet can be obtained by calling the *"getSheet()"* function and passing to the Class *Sheet* object.

Step 3. Access the Cells. The access of Cells can be obtained by calling the "*getCell(i, j)"* function and passing to the Class *NumberCell* object, where *i* stands for Column index and *j* stands for Row index in *getCell(i, j)* starting from 0. After

obtaining the access of Cells, the value can be retrieved by calling the "*getValue()*" function.

The Interface sample codes between Java and Excel Spreadsheet in Sort, Shortest Path, and TSP application examples are shown in the figures below.

```
// Interface between Java and Excel Spreadsheet: Read data from Excel //
// Obtain the Access of Workbook //
Workbook workbook =null;
try
{
    workbook=Workbook.getWorkbook(new File(path));
}
catch(Exception e)
{
    System.out.println(e);
}
// Obtain the Access of Worksheet //
Sheet sheet=workbook.getSheet(0);
// Obtain the Access of Cells //
NumberCell cell=null;
for (int i=0; i<n; i++)
{
    cell=(NumberCell)sheet.getCell(0,i+1);
    arr[i]=(int)cell.getValue();
}
```

Figure A.16 Interface between Java and Excel Spreadsheet in Sort implementation: Read data

```
// Interface between Java and Excel Spreadsheet: write data to Excel //
try
{
    // Obtain the Access of Workbook //
    WritableWorkbook writebook=Workbook.createWorkbook(new File(path),workbook);
    // Obtain the Access of Worksheet //
    WritableSheet writesheet=writebook.getSheet(0);
    // Obtain the Access of Cells //
    jxl.write.Number item=null;
    for (int i=0; i<n; i++)
    {
        item=new jxl.write.Number(1,i+1,a[i]);
        writesheet.addCell(item);
    }
    writebook.write();
    writebook.close();
}
catch (Exception e)
{
    System.out.println(e);
}
```

Figure A.17 Interface between Java and Excel Spreadsheet in Sort implementation: Write result

```
// Interface between Java and Excel Spreadsheet: Read data from Excel //
Workbook workbook=null;
try
{
    // Obtain Access of Workbook //
    workbook=Workbook.getWorkbook(new File("D:\\ShortestPath.xls"));
}
catch(Exception e){ System.out.println(e); }
// Obtain Access of Worksheet //
Sheet sheet=workbook.getSheet("Input");
int row=sheet.getRows();
// Obtain Access of Cells //
for (int i=0;i<row-1;i++)
{
    int p,q;
    double t;
    NumberCell cell1=(NumberCell)sheet.getCell(0,i+1);
    NumberCell cell2=(NumberCell)sheet.getCell(1,i+1);
    NumberCell cell3=(NumberCell)sheet.getCell(2,i+1);
    p=(int)cell1.getValue();
    q=(int)cell2.getValue();
    t=cell3.getValue();
    graph[p][q]=t;
    graph[q][p]=t;
}
```

Figure A.18 Interface between Java and Excel Spreadsheet in Shortest Path implementation: Read data

```
// Interface between Java and Excel Spreadsheet: write data to Excel //
try{
    // Obtain Access of Workbook //
    WritableWorkbook writebook=Workbook.createWorkbook(new File("D:\\ShortestPath.xls"),workbook);
    // Obtain Access of Worksheet //
    WritableSheet writesheet=writebook.getSheet("Output");
    // Access Cells //
    jxl.write.Number item=null;
    for (int i=0; i<n; i++){
        item=new jxl.write.Number(0,i+1,i);
        writesheet.addCell(item);
    }
    for (int i=0; i<n; i++){
        item=new jxl.write.Number(1,i+1,weight[i]);
        writesheet.addCell(item);
    }
    for (int i=0; i<n; i++){
        for (int j=0; j<c[i]; j++){
            item=new jxl.write.Number(j+2,i+1,getpath[i][j]);
            writesheet.addCell(item);
        }
    }
    writebook.write();
    writebook.close();
}
catch (Exception e){
    System.out.println(e);
}
```

Figure A.19 Interface between Java and Excel Spreadsheet in Shortest Path implementation: Write result

```
// Interface between Java and Excel Spreadsheet: Read data from Excel //
Workbook workbook =null;
try {
    // Obtain the Access of Workbook //
    workbook=Workbook.getWorkbook(new File("D:\\TSP.xls"));
}
catch(Exception e){
    System.out.println(e);
}
// Obtain the Access of Worksheet //
Sheet sheet=workbook.getSheet("Input");
// Access Cells //
NumberCell cell1,cell2=null;
for (int i=0; i<n; i++) {
    cell1=(NumberCell)sheet.getCell(0,i+1);
    cell2=(NumberCell)sheet.getCell(1,i+1);
    point[i][0]=(double)cell1.getValue();
    point[i][1]=(double)cell2.getValue();
}
```

Figure A.20 Interface between Java and Excel Spreadsheet in TSP implementation: Read data

```
// Interface between Java and Excel Spreadsheet: write data to Excel //
try{
    // Obtain the Access of Workbook //
    WritableWorkbook writebook=Workbook.createWorkbook(new File("D:\\TSP.xls"),workbook);
    // Obtain the Access of Worksheet //
    WritableSheet writesheet=writebook.getSheet("Output");
    // Access the Cells //
    jxl.write.Number item=null;
    for (int i=0; i<n; i++){
        item=new jxl.write.Number(1,i+1,Route[i]);
        writesheet.addCell(item);
    }
    item=new jxl.write.Number(1,n,0);
    writesheet.addCell(item);
    item=new jxl.write.Number(0,1,TotalDistance);
    writesheet.addCell(item);
    writebook.write();
    writebook.close();
}
catch (Exception e){
    System.out.println(e);
}
```

Figure A.21 Interface between Java and Excel Spreadsheet in TSP implementation: Write result

c) Construct Algorithm Computing codes

Through the Interface between Java and Excel Spreadsheet, the Input data can be read, and we can next construct the algorithm's structure to compute and obtain the result.

(6) Export Java project into a JAR file

Right-Click Java project → Select **Java** → Select **Runnable JAR file** → Click **Next** → Specify **Launch configuration** → Specify **Export destination** → Click **Finish**. The specified Java Runnable JAR file will be generated at the specified destination.

(7) Run the application in Excel Spreadsheet

In order to run the Java JAR file from Excel Spreadsheet, we will create a macro to reference the Java JAR file and run the Macro to trigger the Java JAR file to run. We will write the codes as shown in the Figure below.

```
Sub JavaSortMacro()

Dim JAR As String
Dim retVal As Variant
JAR = """D:\Excel\Java\Java_SORT\java_sort.jar"""
retVal = Shell("C:\Windows\System32\java.exe -jar " & JAR, vbNormalFocus)

End Sub
```

Figure A.22 Excel Macro to trigger Java run in Sort implementation

Hence, the flow of steps is Create an Excel Macro-Enabled Workbook → Enter the **VBE** from Developer Ribbon → Insert a **Module** to create a Macro → Reference the Java JAR file inside the Macro → Run the Macro to trigger the Java JAR file to run.

A.3 HYBRID PROGRAMMING METHOD ON EXCEL

A.3.1 VBA Call C++ DLL Routine

(1) Generate an Input file in Microsoft Excel, Save as Excel Macro-Enabled Workbook, such as "Input.xlsm" file.

(2) Build up the Interface between VBA and Excel Spreadsheet

Firstly, we will build up the Interface between VBA and Excel spreadsheet to read and write data on Excel spreadsheet.

a) Open Visual Basic Editor.

In Excel 2003 and earlier, click **Tools → Macro → Visual Basic Editor** menu item to get into the VBE.

In Excel 2007, Click **Office → Excel Options → Popular tab →** Select **"Show Developer tab in the Ribbon"** will make the Developer ribbon visible. Then click **Developer Ribbon →Visual Basic** button to get into the VBE.

For a quicker access, Press **Alt+F11**.

b) Insert a Module

To insert a module, click the **Insert → Module** menu item.

c) Construct the Interface between VBA and Excel Spreadsheet

We can follow the same structure and steps in VBA Routine on Excel to construct the Interface between VBA and Excel spreadsheet to read and write data on Excel spreadsheet.

(3) Under Microsoft Visual Studio C++ 2010, **Create a new DLL file**

Click the menu item **File → New → Project →** Select **Win32 Console Application → Enter Project name →** Click **OK → Next →** Select **DLL → Finish**.

(4) Build up the Interface between VBA and C++ DLL

a) Define the Interface Function in C++ DLL

The Interface Function should contain two types of arguments, Input arguments and Output arguments, to receive the Input data and pass the Output result repectively. This defined function will be the Interface between VBA and C++ DLL. The Input data enters into this function from VBA, and after computation, the Output result goes out from this function back into VBA. Usually, pointers are used to transfer values between VBA and C++ DLL.

The format to define the Interface function in C++ DLL is shown below.

<*return type*> _stdcall <*function name*> *(function Auguments*)

An example is as follows:

```
double _stdcall sort (double *Input, int n, double *Output)
```

b) Construct the Interface between VBA and C++ DLL in Interface Function

As in the previous step, the data can be exchanged between VBA and C++ DLL through the Interface function arguments. For single value data, the data can be transferred between VBA and C++ DLL by value. For data Array and Matrix, the data value will be transferred between VBA and C++ DLL by address. As data Array and Matrix are consecutively stored in VBA, we can use pointers to reference the address and access the value.

The Library codes of the Interface between VBA and C++ DLL in Sort, Shortest Path, and TSP application examples are shown in the figures below. The Library codes of the Interface between VBA and C++ DLL in VRP spreadsheet application can be found in APPENDIX C.

```
// Interface between VBA and C++ DLL //
// inArr, nCount is the Input arguements; outArr is the Output argument //
double _stdcall sort(double *inArr,int nCount,double *outArr)
{
  // obtain single data value //
  int num=nCount;
  // Use pointers to access address to obtain Input data Array //
  double *arr=new double [nCount];
  for (int i=0; i<nCount; i++)
  {
    arr[i]=inArr[i];
  }

  // arr[] will be sorted through Algorithm Computing //

  // Use pointers to access address to transfer Output result Array //
  for (int i=0; i<nCount; i++)
  {
    outArr[i]=arr[i];
  }
```

Figure A.23 Interface Function between VBA and C++ DLL in Sort implementation

```
// Interface between VBA and C++ DLL //
// graphadj, nDimension are the Input arguments //
// outSPath, outCount, outWeight are the Output arguments //
double _stdcall dijkstra(double *graphadj, int nDimension, double *outSPath,
                         int *outCount, double *outWeight)
{
  // Obtain single data value //
  int n=nDimension;
  // Use pointers to access address to obtain Input data Matrix //
  for (int j=0; j<n; j++)
    for (int k=0; k<n; k++)
      // Data are consecutively stored in VBA //
      graph[k][j]=graphadj[n*j+k];

  // Result are obtained through Algorithm Computing //

  // Use pointers to access address to transfer Output result Array //
  for (int j=0; j<n; j++){
    outCount[j]=c[j];
    outWeight[j]=weight[j];
    for (int k=0; k<n; k++)
      // Data are consecutively stored in VBA //
      outSPath[n*j+k]=getpath[k][j];
  }
```

Figure A.24 Interface Function between VBA and C++ DLL in Shortest Path implementation

```
// Interface between VBA and C++ DLL //
// InputPointx, InputPointy, nCustomer are the Input arguments //
// outRoutePath, outTotalDistance are the Output arguments //
double _stdcall TspOpt(double* InputPointx, double* InputPointy, int nCustomer,
                       int* outRoutePath, double* outTotalDistance)
{
  // Obtain single data value //
  n=nCustomer;
  // Use pointers to access address to obtain Input data Array //
  for (int j=0; j<n; j++)
  {
    pointx[j]=InputPointx[j];
    pointy[j]=InputPointy[j];
  }

  // Result are obtained through Algorithm Computing //

  // Use pointers to access address to transfer Output result Array //
  for (int j=0; j<n; j++)
    outRoutePath[j]=Route[j];
  // Use pointers to access address to transfer Output single value result //
  *outTotalDistance=TotalDistance;
```

Figure A.25 Interface Function between VBA and C++ DLL in TSP implementation

c) Construct Algorithm Computing codes

Through the Interface, the Input data can be read, and we can next construct the algorithm's structure to compute and obtain the result.

d) Export the Interface Function

Add Module-Definition file (.def) into the C++ DLL project to Export the Interface function to be used by VBA.

When creating the C++ DLL project, a ".cpp" file with the same name as the project name will be obtained. Create a ".def" file with the same name as the ".cpp" file to export Functions defined in the ".cpp" file.

Click **Project → Add New Item → Module-Definition file (.def) → Enter name → Add**.

Inside the ".def" file, functions can be exported using the format shown below:

LIBRARY <*Project Name*>

EXPORTS

Function Name @1

...

Function Name @3

An example is shown in the figure below.

```
LIBRARY sort

EXPORTS
sort @1
```

Figure A.26 Export Interface Function in C++ DLL in Sort implementation

e) Build and Generate C++ DLL file

Click the menu item **Build → Build Solution** (F7), and the C++ DLL file will be generated in the Folder of the C++ DLL project.

f) Declare DLL Exported Function in VBA

After Exporting the Interface Function and Generating the C++ DLL file, we can declare the function in VBA by specifying the name of the function and the location of the C++ DLL file. The function return value type and argument type must be matched to the definition in the C++ DLL file.

For data Array and Matrix, the arguments will be declared as "*ByRef*" to pass the data between VBA and C++ DLL by address. For single value data, the arguments will be declared as "*ByVal*" to pass the data between VBA and C++ DLL by value.

For example, the following figures show what can be written to declare the Sort function exported in the previous steps.

```
Option Explicit
Public Declare Function sort Lib "D:\Excel\VBA Call DLL\DLL_SORT\sort.dll" _
                                 (ByRef inarr As Double, ByVal nCount As Long, _
                                 ByRef outarr As Double) As Double
```

Figure A.27 Declare Exported C++ DLL Function in VBA in Sort implementation

```
Option Explicit
Public Declare Function dijkstra Lib "D:\Excel\VBA Call DLL\DLL_ShortestPath\shortestpath.dll" _
                                    (ByRef graphadj As Double, ByVal nDimension As Long, _
                                    ByRef outSPath As Double, ByRef outCount As Long, _
                                    ByRef outWeight As Double) As Double
```

Figure A.28 Declare Exported C++ DLL Function in VBA in Shortest Path implementation

```
Option Explicit
Public Declare Function TspOpt Lib "D:\Excel\VBA Call DLL\DLL_TSP\TSP.dll" _
                                (ByRef InputPointx As Double, ByRef InputPointy As Double, _
                                ByVal nCustomer As Long, ByRef outRoutePath As Long, _
                                ByRef TotalDistance As Double) As Double
```

Figure A.29 Declare Exported C++ DLL Function in VBA in TSP implementation

(5) Call the Declared function in VBA

After declaring the function in VBA, we can use it as a self-defined function in VBA. Hence, we can read Input data from Excel spreadsheet through the Interface between VBA and Excel, and pass the data to C++ DLL by calling the declared C++ DLL function, and obtain the result through the Interface between VBA and C++ DLL. Then, the result will be written back to Excel spreadsheet through the Interface between VBA and Excel.

When calling the declared function in VBA, if the data is an Array and arguments are passed "*ByRef*", we just need to specify the first element's address in the function arguments. Examples are shown in the following figures.

```
retVal = sort(inarr(0), nCount, outarr(0))
```

Figure A.30 Call Declared C++ DLL function in VBA in Sort implementation

```
retVal = dijkstra(graphadj(0, 0), nDimension, outSPath(0, 0), outCount(0), outWeight(0))
```

Figure A.31 Call Declared C++ DLL function in VBA in Shortest Path implementation

```
retVal = TspOpt(InPointx(0), InPointy(0), nCustomer, outRoutePath(0), outTotalDistance)
```

Figure A.32 Call Declared C++ DLL function in VBA in TSP implementation

(6) Run the Application in VBA

After bridging Excel spreadsheet with VBA, bridging VBA with C++ DLL, and constructing the algorithm computing function inside C++ DLL, we can run the application in VBA and Excel spreadsheet, and complete building applications on Excel spreadsheet using VBA call C++ DLL method.

APPENDIX B

Routines of Different Methods to Build Applications on Calc Spreadsheet

B.1 INTERNAL PROGRAMMING METHOD ON CALC

B.1.1 OOOBasic Routine

(1) Generate an Input file in OpenOffice.org Calc, Save as ODF Spreadsheet file, such as "Input.ods" file.

(2) Add OpenOffice.org Macro.

In OpenOffice.org 3.3.0, Click **Tools → Macros → Organize Macros → OpenOffice.org Basic (Alt + F11) → New** to create a new module and get you into the OpenOffice.org Basic Editor.

(3) Write the codes of Application

a) Construct the Interface between OOO Basic and Calc spreadsheet (Reading & Writing)

To build up the Interface between OOO Basic and Calc spreadsheet, we need to obtain three levels of access on Calc spreadsheet step by step.

Step 1. Obtain the Access of SpreadsheetDocument. The access of SpreadsheetDocument can be obtained by calling the *"LoadComponentFromURL()"* function and declaring an object type variable to receive it. If the SpreadsheetDocument has already been opened and the application is built directly on it, then we can simply declare an object and use *"ThisComponent"* to obtain the access of this SpreadsheetDocument.

Step 2. Obtain the Access of Sheets. The access of Sheets can be obtained by calling the *"Sheets"* and *"getByName()"* functions and received by declaring an object type variable.

Step 3. Obtain the Access of Cells. The access of Cells can be specified by calling the *"getCellByPosition(i, j)"* function and received by declaring an object type variable, where *i* stands for the Column index and *j* stands for the Row index starting from 0. The data value stored in *Cell (i, j)* can be retrieved by using the *"Value"* property.

The Interface sample codes between OOO Basic and Calc spreadsheet in Sort, Shortest Path, and TSP application examples are shown in the figures below.

```
Dim Doc as Object
Dim sheet as Object
Dim cell as Object
Dim cellrange as Object
Dim Inputdata as Variant
Dim i As Long
Dim n As Long

'// Access the Document level //
Doc = ThisComponent
'// Access the Sheets level //
sheet = Doc.Sheets.getByName("Sheet1")
'// Access the Cells level //
cell=sheet.getCellByPosition(3,1)
'// Retrieve the value //
n=cell.Value
cellrange=sheet.getCellRangeByPosition(0,1,0,size)
Inputdata=cellrange.Data
For i = 0 To n-1
   arr(i)=data(i)(0)
Next i
```

Figure B.1 Interface between OOO Basic and Calc spreadsheet in Sort implementation: Read data

```
Dim result as Variant
'// Access the Document level //
Doc = ThisComponent
'// Access the Sheets level //
sheet = Doc.Sheets.getByName("Sheet1")
cellrange=sheet.getCellRangeByPosition(1,1,1,size)
result = cellrange.Data
For i = 0 To n-1
   result(i)(0) = arr(i)
Next i
cellrange.Data = result
```

Figure B.2 Interface between OOO Basic and Calc spreadsheet in Sort implementation: Write result

```
Dim Doc as Object
Dim sheet as Object
Dim cell as Object
Dim n As Long
Dim maxRows As Integer
' Access the SpreadsheetDocument
Doc=ThisComponent
' Access the Sheets
sheet=Doc.Sheets.getByName("Input")
cursor=sheet.createCursor
cursor.gotoEndOfUsedArea(True)
maxRows = cursor.Rows.Count
' Access the Cells
For i = 1 To maxRows-1
    cell=sheet.getCellByPosition(0,i)
    p = cell.Value
    cell=sheet.getCellByPosition(1,i)
    q = cell.Value
    cell=sheet.getCellByPosition(2,i)
    t = cell.Value
    graph(p, q) = t
    graph(q, p) = t
Next i
```

Figure B.3 Interface between OOO Basic and Calc spreadsheet in Shortest Path implementation: Read data

```
' Access the SpreadsheetDocument
Doc=ThisComponent
' Access the Sheets
sheet=Doc.Sheets.getByName("Output")
' Access the Cells
For j = 1 To n
     cell=sheet.getCellByPosition(1,j)
     cell.Value = weight(j - 1)
     cell=sheet.getCellByPosition(0,j)
     cell.Value = j - 1
Next j
For i = 0 To n - 1
       For j = 0 To count(i) - 1
            cell=sheet.getCellByPosition(j+2,i+1)
            cell.Value = getpath(i, j)
       Next j
Next i
```

Figure B.4 Interface between OOO Basic and Calc spreadsheet in Shortest Path implementation: Write result

```
Dim Doc as Object
Dim sheet as Object
Dim cell as Object
Dim n as Long

' Access the SpreadsheetDocument
Doc=ThisComponent
' Access the Sheets
sheet=Doc.Sheets.getByName("Input")
' Access the Cells
For i = 1 To n
   cell=sheet.getCellByPosition(0,i)
   point(i - 1, 0) = cell.Value
   cell=sheet.getCellByPosition(1,i)
   point(i - 1, 1) = cell.Value
Next i
```

Figure B.5 Interface between OOO Basic and Calc spreadsheet in TSP implementation: Read data

```
' Access the SpreadsheetDocument
Doc=ThisComponent
' Access the Sheets
sheet=Doc.Sheets.getByName("Output")
' Access the Cells
For i = 1 To n
   cell=sheet.getCellByPosition(1,i)
   cell.Value = Route(i - 1)
Next i
cell=sheet.getCellByPosition(1 , n)
cell.Value = 0
cell=sheet.getCellByPosition(0, 1)
cell.Value = TotalDistance
```

Figure B.6 Interface between OOO Basic and Calc spreadsheet in TSP implementation: Write result

b) Construct Algorithm Computing Codes

Through the Interface between OOO Basic and Calc spreadsheet, the Input data can be read, and we can next construct the algorithm's structure to compute and obtain the result.

(4) Run the application in Calc spreadsheet

After finishing all the above steps, we can run the application in Calc spreadsheet to complete the whole process of building applications on Calc spreadsheet using the OOO Basic method.

B.2 EXTERNAL PROGRAMMING METHOD ON CALC

B.2.1 Java Routine

(1) Generate an Input file in OpenOffice.org Calc, Save as ODF spreadsheet file, such as "Input.ods" file.

(2) Under Eclipse IDE for Java Developers, **Create a new Java project**

Click the menu item **File → New → Java Project → Enter Project name → Next → Finish**.

(3) Add Open-source Library into the Java project

In Java Routine on Calc spreadsheet, the Interface between Java and Calc spreadsheet, such as reading and writing on Calc, is accomplished by using an open source library called ODFDOM and SimpleJavaAPI downloaded from the Internet.

Click the menu item **Project → Properties → Java Build Path →** Select the label **Libraries →** Click **Add External JARs →** Select .**jar file** listed below **→ OK**.

The open-source library files included are listed below:

- odfdom-java-0.8.7.jar
- simple-odf-0.6.jar
- xerces-2_11_0\resolver.jar
- xerces-2_11_0\serializer.jar
- xerces-2_11_0\xercesImpl.jar
- xerces-2_11_0\xercesSamples.jar
- xerces-2_11_0\xml-apis.jar

(4) Create a new Class (.java) file

The Class file created will be the place to develop smart applications, and the number of Class files needed will depend on the application's requirement.

Right Click the Java Project **→ New → Class → Enter Class name →** Select **"public static void main(String[] args)"** (Optional) **→ Finish**.

(5) Write the Codes of Application

a) Import Open-Source Library Classes

We will use the Classes and Functions defined in SimpleJavaAPI Library to build up the Interface between Java and Calc spreadsheet. Hence, we have to import the SimpleJavaAPI Library Classes first. The other import Libraries needed will depend on the application's requirement. This is shown in the Figure below.

```
import org.odftoolkit.simple.SpreadsheetDocument;
import org.odftoolkit.simple.table.Table;
import org.odftoolkit.simple.table.Cell;
```

Figure B.7 Java on Calc spreadsheet: Import Classes

b) Construct the Interface between Java and Calc spreadsheet (Reading & Writing)

To build up the Interface with reading and writing on Calc spreadsheet, we need to obtain three levels of access on Calc step by step.

Step 1. Access the SpreadsheetDocument. The access of SpreadsheetDocument can be obtained by calling the *"loadDocument( )"* function and passing to an object with *"SpreadsheetDocument"* Class type.

Step 2. Access the Sheets. The access of Sheets can be obtained by using the SpreadsheetDocument object to call the *"getSheetByIndex( )"* function to retrieve Sheet by index and passing to an object with *"Table"* Class type.

Step 3. Access the Cells. The access of Cells can be obtained by using the Table object "*sheet*" defined above to call the *"getCellByPosition(i, j)"* function, where *i* stands for the Column index and *j* stands for the Row index starting from 0, and then passing to an object defined with *"Cell"* Class type. The data value stored in *Cells(i, j)* can be retrieved by calling functions such as *"getDoubleValue()"*, *"getIntegerValue()"*, and *"getStringValue()"*, etc.

The Library codes of the Interface between Java and Calc spreadsheet in Sort, Shortest Path, and TSP application examples are shown in the figures below.

```
// Interface between Java and Calc Spreadsheet: Read Input data //
SpreadsheetDocument workbook =null;
try
{
    // Access SpreadsheetDocument //
    workbook=SpreadsheetDocument.loadDocument("D:\\Sort.ods");
}
catch(Exception e)
{
    System.out.println(e);
}
// Access Sheets //
Table sheet=workbook.getSheetByIndex(0);
Cell cell=null;
for (int i=0; i<n; i++)
{
    // Access Cells //
    cell=sheet.getCellByPosition(0,i+1);
    a[i]=cell.getDoubleValue();
}
```

Figure B.8 Interface between Java and Calc spreadsheet in Sort implementation: Read data

```
// Interface between Java and Calc Spreadsheet: write result //
SpreadsheetDocument workbook_w =null;
try
{
    // Access SpreadsheetDocument //
    workbook_w=SpreadsheetDocument.loadDocument("D:\\Sort.ods");
}
catch(Exception e){ System.out.println(e); }
// Access Sheets //
Table sheet_w=workbook_w.getSheetByIndex(0);
Cell cell_w=null;
for (int i=0;i<n;i++)
{
    // Access Cells //
    cell_w=sheet_w.getCellByPosition(1,i+1);
    cell_w.setDoubleValue(a[i]);
}
try
{
    workbook_w.save("D:\\Sort.ods");
}
catch(Exception e){ System.out.println(e); }
```

Figure B.9 Interface between Java and Calc spreadsheet in Sort implementation: Write result

```
// Interface between Java and Calc Spreadsheet: Read data from Calc //
// Obtain the Access of Workbook //
SpreadsheetDocument workbook=null;
try{
    workbook=SpreadsheetDocument.loadDocument("D:\\ShortestPath.ods");
}
catch(Exception e){
    System.out.println(e);
}
// Obtain the Access of Sheets //
Table sheet=workbook.getSheetByName("Input");
// Obtain the Access of Cells //
for (int i=0;i<Maxrow-1;i++){
    int p,q;
    double t;
    Cell cell1=sheet.getCellByPosition(0,i+1);
    Cell cell2=sheet.getCellByPosition(1,i+1);
    Cell cell3=sheet.getCellByPosition(2,i+1);
    p=(int)(double)cell1.getDoubleValue();
    q=(int)(double)cell2.getDoubleValue();
    t=cell3.getDoubleValue();
    graph[p][q]=t;
    graph[q][p]=t;
}
```

Figure B.10 Interface between Java and Calc spreadsheet in Shortest Path implementation: Read data

```
// Interface between Java and Calc Spreadsheet: write result to Calc //
// Obtain the Access of SpreadsheetDocument //
try{
    workbook=SpreadsheetDocument.loadDocument("D:\\ShortestPath.ods");
}
catch(Exception e){
    System.out.println(e);
}
// Obtain the Access of Sheets //
Table sheet2=workbook.getSheetByName("Output");
// Access Cells //
for (int i=0; i<n; i++){
    Cell cell1=sheet2.getCellByPosition(0,i+1);
    cell1.setDoubleValue((double)i);
    Cell cell2=sheet2.getCellByPosition(1,i+1);
    cell2.setDoubleValue(weight[i]);

    for (int j=0; j<c[i]; j++){
        Cell cell=sheet2.getCellByPosition(j+2,i+1);
        cell.setDoubleValue((double)getpath[i][j]);
    }
}
try{
    workbook.save("D:\\ShortestPath.ods");
}
catch (Exception e){
    System.out.println(e);
}
```

Figure B.11 Interface between Java and Calc spreadsheet in Shortest Path implementation: Write result

```
// Interface between Java and Calc Spreadsheet: Read data from Calc //
// Obtain the Access of SpreadsheetDocument //
SpreadsheetDocument workbook=null;
try {
    workbook=SpreadsheetDocument.loadDocument("D:\\TSP.ods");
}
catch(Exception e){
    System.out.println(e);
}
// Obtain the Access of Sheets //
Table sheet=workbook.getSheetByName("Input");
// Obtain the Access of Cells //
Cell cell1,cell2=null;
for (int i=0; i<n; i++)
{
    cell1=sheet.getCellByPosition(0,i+1);
    cell2=sheet.getCellByPosition(1,i+1);
    point[i][0]=(double)cell1.getDoubleValue();
    point[i][1]=(double)cell2.getDoubleValue();
}
```

Figure B.12 Interface between Java and Calc spreadsheet in TSP implementation: Read data

```
// Interface between Java and Calc Spreadsheet: write result to Calc //
// Obtain the Access of SpreadsheetDocument //
try {
    workbook=SpreadsheetDocument.loadDocument("D:\\TSP.ods");
}
catch(Exception e){
    System.out.println(e);
}
// Obtain the Access of Sheets //
Table sheet2=workbook.getSheetByName("Output");
// Access the Cells //
for (int i=0; i<n; i++)
{
    Cell cell=sheet2.getCellByPosition(1,i+1);
    cell.setDoubleValue((double)Route[i]);
}
Cell cell=sheet2.getCellByPosition(0,1);
cell.setDoubleValue(TotalDistance);
try{
    workbook.save("D:\\TSP.ods");
}
catch (Exception e){
    System.out.println(e);
}
```

Figure B.13 Interface between Java and Calc spreadsheet in TSP implementation: Write result

c) Construct Algorithm Computing codes

Through the Interface between Java and Calc spreadsheet, the Input data can be read, and we can next construct the algorithm's structure to compute and obtain the result.

(6) Export Java project into a JAR file

Right Click Java project → Select **Java** → Select **Runnable JAR file** → Click **Next** → Specify **Launch configuration** → Specify **Export destination** → Click **Finish**. The specified Java Run-able JAR file will be generated at the specified destination.

(7) Run the application in Calc spreadsheet

In order to run the Java JAR file in Calc spreadsheet, we will create a macro in Calc to reference the Java JAR file and run the Macro to trigger the Java JAR file to run.

Firstly, we will create a batch file (.bat) to write the command line which will trigger the Java Runnable JAR file to run, such as

START java.exe -jar *"D:\Calc\Java\Calc_Java_SORT\OpenOffice_sort.jar"*

Then, we will write the Macro in Calc spreadsheet to run the batch file (.bat) to trigger the Java JAR file to run as shown in the Figure below.

```
Sub Calc_Java_SORT

Dim retVal As Variant
retVal = Shell(ConvertToURL("D:\Calc\Java\Calc_Java_SORT\sort.bat"),1)

End Sub
```

Figure B.14 Calc Macro to trigger Java JAR run in Sort implementation

Hence, the flow of steps is Create a batch file (.bat) to trigger Java JAR file to run → Add OpenOffice.org Macro → Reference the batch file (.bat) to run → Run the Macro to trigger the Java JAR file to run.

After finishing all the above steps, we can complete the process of building applications on Calc spreadsheet using the Java method.

APPENDIX C

Library Codes of Interface between VBA and C++ DLL in VRP Spreadsheet Application

C.1 The Interface function VRPprocess with Input and Output arguments to transfer data between VBA and C++ DLL

```
double _stdcall VRPprocess(int n_v, int n_caplimit, int n_customer, int * pPosX, int* pPosY, \
                           int* pDemand, int* pReadyTime, int* pLateTime, int* pServiceTime, \
                        int* n_VehiclesUsed, int* n_CustomerServed, int* n_CustomerNotServed, \
                              double* TotalDistance, int* TotalLoad, int* nCustomerInRoute, \
                                    int* nCustomerNumber, int* nRouteArrivalTime)
{
        // Creat Class object;
        CVRPTW VRPTW;
        // Receive data from the argument variables
        VRPTW.TransferInput(n_v, n_caplimit, n_customer, pPosX, pPosY, \
                                          pDemand, pReadyTime, pLateTime, pServiceTime);
        // Optimize and obtain the solution
        VRPTW.Optimise();
        // Transfer result to the argument variables
        VRPTW.ReportOutput(n_VehiclesUsed, n_CustomerServed, n_CustomerNotServed, \
                                                              TotalDistance, TotalLoad, \
                                    nCustomerInRoute, nCustomerNumber, nRouteArrivalTime);
        // Free the memory and space
        VRPTW.OnFree();
        return 0;
}
```

C.2 Sub-function *TransferInput*() with Input arguments in VRPprocess to transfer Input data from C++ DLL to VBA

```
void CVRPTW::TransferInput(int n_v, int n_caplimit, int n_customer, int * pPosX,
                           int* pPosY, int* pDemand, int* pReadyTime, int* pLateTime, \
                                                                    int* pServiceTime)
{
        // Create instance for route optimisation
        pVehicleRoute = new CVehicleRoute(&TestCase);
        pRouteOptimisation = new CRouteOptimisation(pVehicleRoute);

        // Transfer Test case data to C++ DLL
        TestCase.LoadData (n_v, n_caplimit, n_customer, pPosX, pPosY, pDemand, \
                                                  pReadyTime, pLateTime, pServiceTime)

        // Initialise route
        pRouteOptimisation->Initialisation();
}
```

C.3 Sub-function *LoadData()* in *TransferInput()* to transfer Input data

```
bool CTestCase::LoadData(int n_v, int n_caplimit, int n_customer, int * pPosX, int* pPosY,
                                                    int* pDemand, int* pReadyTime, int* pLateTime,
                                                                                    int* pServiceTime)
{
        // Clear pointer
        if(pCustomerPosX)
                delete [] pCustomerPosX;
        if(pCustomerPosY)
                delete [] pCustomerPosY;
        if(pCustomerDemand)
                delete [] pCustomerDemand;
        if(pCustomerReadyTime)
                delete [] pCustomerReadyTime;
        if(pCustomerLateTime)
                delete [] pCustomerLateTime;
        if(pCustomerServiceTime)
                delete [] pCustomerServiceTime;
        if(pCustomerValue)
                delete [] pCustomerValue;
        if(pDistanceMatrix)
        {
                for(int i=0;i<=nNumberOfCustomers;i++)
                        delete [] pDistanceMatrix[i];
                delete [] pDistanceMatrix;
        }

         //Pass data to C++ DLL from VBA through function arguments by address or by
        value
        nNumberOfVehicles=n_v;
        nCapacityLimit=n_caplimit;
        nNumberOfCustomers=n_customer;
        pCustomerPosX=new int[nNumberOfCustomers+1];
        pCustomerPosY=new int[nNumberOfCustomers+1];
        pCustomerDemand=new int[nNumberOfCustomers+1];
        pCustomerReadyTime=new int[nNumberOfCustomers+1];
        pCustomerLateTime=new int[nNumberOfCustomers+1];
        pCustomerServiceTime=new int[nNumberOfCustomers+1];
        pCustomerValue=new int[nNumberOfCustomers+1];

        for(int i=0;i<=nNumberOfCustomers;i++)
        {
                pCustomerPosX[i]=pPosX[i];
                pCustomerPosX[i]*=DISTANCETIMESCALE;
                pCustomerPosY[i]=pPosY[i];
                pCustomerPosY[i]*=DISTANCETIMESCALE;
                pCustomerDemand[i]=pDemand[i];
                pCustomerReadyTime[i]=pReadyTime[i];
                pCustomerReadyTime[i]*=DISTANCETIMESCALE;
                pCustomerLateTime[i]=pLateTime[i];
                pCustomerLateTime[i]*=DISTANCETIMESCALE;
                pCustomerServiceTime[i]=pServiceTime[i];
                pCustomerServiceTime[i]*=DISTANCETIMESCALE;
```

```
                pCustomerValue[i] = pCustomerDemand[i];
        }

        // Allocate memory for Distance Matrix
        // And calculate distance
        pDistanceMatrix = new unsigned int*[nNumberOfCustomers+1];
        double x,y;
        for(int i=0;i<=nNumberOfCustomers;i++)
        {
                pDistanceMatrix[i] = new unsigned int[nNumberOfCustomers+1];
                for(int j=0;j<=nNumberOfCustomers;j++)
                {
                        x = pCustomerPosX[i] - pCustomerPosX[j];
                        y = pCustomerPosY[i] - pCustomerPosY[j];
                        pDistanceMatrix[i][j] = (unsigned int)(sqrt(x*x + y*y)+0.5);
                }
        }
        return true;
}
```

C.4 Sub-function *ReportOutput()* with Output arguments in *VRPprocess()* to transfer Output result from C++ DLL back to VBA

```
void CVRPTW::ReportOutput(int* n_VehiclesUsed,int* n_CustomerServed, \
                          int* n_CustomerNotServed, double* TotalDistance, int* TotalLoad, \
                  int* nCustomerInRoute, int* nCustomerNumber, int* nRouteArrivalTime)
{
        // Use pointers to report results of static values to VBA by address through function
        arguments
        *n_VehiclesUsed= pVehicleRoute->GetNumVehicleUsed();
        *n_CustomerServed=pVehicleRoute->GetCustomerInserted();
        *n_CustomerNotServed=pVehicleRoute->GetHoldCount();
        *TotalDistance=pVehicleRoute->GetTotalDistance();
        *TotalLoad=pVehicleRoute->GetTotalCustomerValue();

        // RouteResult function handles the dynamic length of route and arrival time result
        array
        pRouteOptimisation->RouteResult(nCustomerInRoute, nCustomerNumber, \
                                                                nRouteArrivalTime);

}
```

C.5 Sub-function *RouteResult()* in *ReportOutput()* to Sync route result with dynamic length information between C++ DLL and VBA

```
    void CRouteOptimisation::RouteResult(int* nCustomerInRoute, int* nCustomerNumber, \
                                                                int* nRouteArrivalTime)
{
        // Use pointers to report routes to VBA by address through function arguments
        // Handle the dynamic length of route and arrival time result array
```

```
        // Define a large matrix with same size in VBA capable to handle the maximum
        dynamic length
        int nDimension = pVehicleRoute->nNumberOfCustomers;
        int* CustomerInRoute = new int [nDimension];
        int ** CustomerID = new int* [nDimension];
        int ** RouteArrivalTime = new int* [nDimension];
        for (int k=0; k < nDimension; k++)
        {
                CustomerID[k]=new int [nDimension];
                RouteArrivalTime[k]=new int [nDimension];
        }

        // Push the dynamic length route and arrival time result array into the large matrix
        for (int j=0;j<pVehicleRoute->nVehicleUsed;j++)
        {
                CustomerInRoute[j] = pVehicleRoute->GetRouteTotalCustomers(j);
                pVehicleRoute->SetRouteDepot(j);
                for(int i=0;i<CustomerInRoute[j];i++)
                {
                        pVehicleRoute->GoNextCustomer(j);
                        CustomerID[j][i] = pVehicleRoute->GetRouteCustomer(j);
                        RouteArrivalTime[j][i]=pVehicleRoute->GetRouteArrivalTime(j);
                }
        }

        // Sync the large matrix in both VBA and C++ DLL to pass the route result to VBA
        for (int k=0; k<nDimension; k++)
                nCustomerInRoute[k]=CustomerInRoute[k];
         for (int j=0; j<nDimension; j++)
                for (int k=0; k<nDimension; k++)
                {
                        nCustomerNumber[nDimension*j+k]=CustomerID[k][j];
                        nRouteArrivalTime[nDimension*j+k]=RouteArrivalTime[k][j];
                }
}
```

APPENDIX D

Performance Comparison between Excel VRPTW Application and C++ Standalone VRPTW Application in 56 Solomon Test Cases

Solomon Test Cases	Total time*		Reading		Algorithm computing		Writing	
	Excel	C++	Excel	C++	Excel	C++	Excel	C++
C101	3.143	2.964	0.009	0.002	2.563	2.955	0.571	0.007
C102	4.536	4.285	0.010	0.002	3.991	4.275	0.535	0.008
C103	6.605	5.191	0.010	0.002	5.957	5.181	0.638	0.008
C104	8.799	8.565	0.014	0.002	8.232	8.555	0.553	0.008
C105	4.293	3.750	0.009	0.002	3.748	3.740	0.536	0.008
C106	4.788	3.939	0.010	0.002	4.173	3.929	0.605	0.008
C107	4.992	4.252	0.008	0.002	4.384	4.242	0.600	0.008
C108	5.945	6.109	0.009	0.002	5.242	6.099	0.694	0.008
C109	8.068	7.097	0.009	0.002	7.291	7.087	0.768	0.008
C201	2.167	1.368	0.009	0.002	1.420	1.358	0.738	0.008
C202	2.869	2.602	0.012	0.002	2.075	2.593	0.782	0.007
C203	4.440	3.383	0.010	0.004	3.614	3.371	0.816	0.008
C204	6.127	4.125	0.012	0.003	5.294	4.114	0.821	0.008
C205	2.496	1.586	0.018	0.002	1.614	1.576	0.864	0.008
C206	2.871	2.005	0.011	0.002	1.983	1.996	0.877	0.007
C207	3.140	2.281	0.011	0.002	2.176	2.272	0.953	0.007
C208	3.471	2.287	0.013	0.003	2.518	2.276	0.940	0.008
R101	7.861	7.683	0.017	0.001	7.410	7.674	0.434	0.008
R102	10.240	10.759	0.010	0.002	9.768	10.750	0.462	0.007
R103	8.804	9.704	0.008	0.002	8.308	9.694	0.488	0.008
R104	9.436	8.448	0.008	0.002	8.871	8.438	0.557	0.008
R105	7.158	6.039	0.012	0.002	6.553	6.029	0.593	0.008
R106	8.228	7.807	0.009	0.002	7.597	7.797	0.622	0.008
R107	6.257	6.164	0.009	0.002	5.593	6.154	0.655	0.008
R108	8.182	5.618	0.012	0.002	7.499	5.608	0.671	0.008
R109	7.545	7.529	0.015	0.001	6.835	7.520	0.695	0.008
R110	7.538	6.211	0.018	0.003	6.800	6.200	0.720	0.008
R111	8.449	9.971	0.010	0.005	7.645	9.958	0.794	0.008
R112	8.729	7.935	0.010	0.003	7.926	7.924	0.793	0.008
R201	3.037	2.899	0.008	0.002	2.617	2.889	0.412	0.008
R202	4.544	4.276	0.008	0.002	4.092	4.266	0.444	0.008
R203	4.817	3.591	0.008	0.002	4.333	3.582	0.476	0.007
R204	7.538	6.727	0.008	0.002	7.010	6.718	0.520	0.007
R205	3.833	2.303	0.008	0.003	3.288	2.292	0.537	0.008
R206	4.490	4.084	0.012	0.002	3.911	4.074	0.567	0.008
R207	6.513	5.471	0.009	0.002	5.898	5.462	0.606	0.007

R208	7.299	3.886	0.009	0.002	6.623	3.876	0.667	0.008
R209	6.476	3.597	0.009	0.003	5.739	3.587	0.728	0.007
R210	4.880	3.578	0.010	0.002	4.078	3.569	0.792	0.007
R211	5.092	4.674	0.010	0.002	4.310	4.664	0.772	0.008
RC101	6.540	7.279	0.010	0.002	5.715	7.270	0.815	0.007
RC102	8.738	6.211	0.011	0.002	7.840	6.201	0.887	0.008
RC103	7.402	8.743	0.010	0.002	6.530	8.734	0.862	0.007
RC104	9.993	6.449	0.011	0.002	9.050	6.439	0.932	0.008
RC105	7.818	8.055	0.010	0.002	6.881	8.046	0.927	0.007
RC106	7.267	7.219	0.011	0.003	6.301	7.208	0.955	0.008
RC107	7.729	7.465	0.012	0.003	6.587	7.454	1.130	0.008
RC108	7.466	7.064	0.010	0.003	6.373	7.053	1.083	0.008
RC201	3.876	2.676	0.010	0.002	2.711	2.667	1.155	0.007
RC202	4.352	4.048	0.011	0.002	3.213	4.039	1.128	0.007
RC203	8.281	5.632	0.010	0.002	7.068	5.623	1.203	0.007
RC204	7.605	4.820	0.010	0.003	6.437	4.809	1.158	0.008
RC205	5.283	3.388	0.009	0.002	4.073	3.379	1.201	0.007
RC206	3.532	3.588	0.009	0.002	2.308	3.579	1.215	0.007
RC207	6.037	5.779	0.009	0.003	4.727	5.769	1.302	0.007
RC208	5.870	4.313	0.009	0.002	4.596	4.304	1.266	0.007

***** *All the running times are measured in Seconds.*