# Time-Cost Optimization of Large-Scale Construction Projects Using Constraint Programming

by

Behrooz Golzarpoor

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Management Sciences

Waterloo, Ontario, Canada, 2012

# Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

# Abstract

Optimization of time and cost in construction projects has been subject to extensive research since the development of the Critical Path Method (CPM). Many researchers have investigated various versions of the well-known Time-Cost Trade-off (TCT) problem including linear, convex, concave, and also the discrete (DTCT) version. Traditional methods in the literature for optimizing time and cost of construction projects range from mathematical methods to evolutionary-based ones, such as genetic algorithms, particle swarm, ant-colony, and leap frog optimization. However, none of the existing research studies has dealt with the optimization of large-scale projects in which any small saving would be significant. Traditional approaches have all been applied to projects of less than 100 activities which are far less than what exists in real-world construction projects. The objective of this study is to utilize recent developments in computation technology and novel optimization techniques such as Constraint Programming (CP) to improve the current limitations in solving large-scale DTCT problems.

Throughout the first part of this research, an Excel-based TCT model has been developed to investigate the performance of traditional optimization methods, such as mathematical programming and genetic algorithms, for solving large TCT problems. The result of several experimentations confirms the inefficiency of traditional methods for optimizing large TCT problems. Subsequently, a TCT model has been developed using Optimization Programming Language (OPL) to implement the Constraint Programming (CP) technique. CP Optimizer of IBM ILOG Optimization Studio has been used to solve the model and to successfully

optimize several projects ranging from a small project of 18 activities to very large projects consisting of more than 10,000 activities. Constraint programming proved to be very efficient in solving large-scale TCT problems, generating substantially better results in terms of solution quality and processing speed.

While traditional optimization methods have been used to optimize projects consisting of less than one hundred activities, constraint programming demonstrated its capability of solving TCT problems comprising of thousands of activities. As such, the developed model represents a significant improvement in optimization of time and cost of large-scale construction projects and can greatly enhance the level of planning and control in such projects.

# Acknowledgements

First and foremost I would like to express my sincere appreciation to my supervisors, Dr. Tarek Hegazy and Dr. Frank Safayeni for their invaluable guidance, inspiration and patience during my master's program. The opportunity of taking part in their remarkable course lectures as well as having their support on my research has been a great honor for me.

I would also like to thank my readers, Dr. Carl T. Haas and Dr. Hossein Abouee, for their encouragement and their insightful comments to improve my thesis.

I am grateful to Mr. Vu Huynh, Systems Administrator, for providing me access to the required computer facilities to perform a major part of my experimentations. The very usefulness of the IBM Academic Initiative program is also acknowledged for offering access to the constraint programming optimization software.

Last but certainly not least, I would like to express my gratitude to Dr. Wail Menesi for his valuable comments and suggestions on my thesis.

*To My Parents*

*And My Wife*

# Table of Contents

# List of Figures

# List of Tables

# Chapter 1
# Introduction

## 1.1 General Introduction

Time and cost are the most important concerns in every construction project (Ng & Zhang, 2008). Time and cost determine the feasibility of a project in the preliminary phase of the project's lifecycle; have a considerable impact on the outcome of the design and the planning phases; and influence the success or failure of the project throughout the subsequent phases of the project.

Since the introduction of Critical Path Method (CPM) in 1950s, CPM has been the basis for project scheduling and calculating the total project time and cost. However, CPM calculates the total project's duration as the sum of the duration of activities on the critical path and, therefore, does not explicitly respect the time limitation (deadline) of the project. To overcome this drawback of the CPM, time-cost trade-off (TCT) analysis was developed. In the TCT analysis, the objective is to reduce the original CPM duration of the project to meet a specific deadline in the least costly way. This can be achieved by the optimum selection of some activities to be performed using faster, and usually more expensive, construction methods (Hegazy & Ayed, 1999).

In general, there is a trade-off between time and cost for completing a task (Figure 1-1); the less expensive the resources, the longer it takes to complete an activity (Feng, Liu, & Burns, 1997). For a project, the total cost is the sum of direct and indirect costs and there is an optimum duration for the least cost (Figure 1-2).

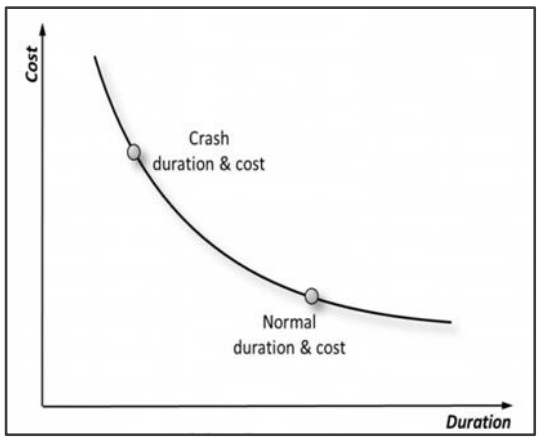


**Figure 1-1 Activity Time-Cost Relation**

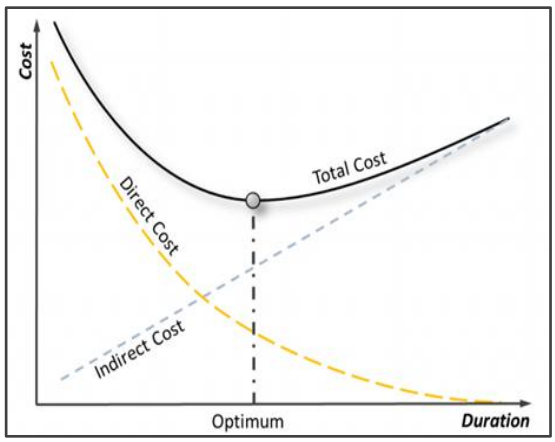


**Figure 1-2 Project Time-Cost Curve**

Typically, activities may have different execution options (modes) that can include possible combinations of: 1) construction methods, which denote possible construction technologies and/or materials; 2) subcontractors' quotes, which represent the proposed duration and cost of performing the activities by subcontractors, 3) crew formations, which symbolize feasible arrangements of construction labor and equipment; and 4) overtime strategies, which define the length and time of work shifts (Tarek Hegazy & Menesi, 2010; El-Rayes & Kandil, 2005).

The selection of any mode of execution for each activity leads to a distinct time and cost for that activity and affects the overall duration and cost for the entire project. The combination of various possible execution modes of activities produces several project plans where each project plan has a unique duration and cost. For large projects, the enumeration of these alternative project plans is computationally hard, particularly because the number of alternatives grows exponentially with the increase in the number of activities of the project. For example, the number of possible alternative project plans for a project consisting of 180 activities, each with only 3 modes of construction can reach $3^{180}$ ($7.6 \times 10^{85}$) (Kandil & El-

Rayes, 2006). The number of possible alternatives increases $3.5x10^9$ times ($2.7x10^{95}$), if the number of activities in the project network rises only to 200 activities.

In the literature, many models have been proposed to solve this combinatorial optimization problem. The modeling techniques range from the heuristic methods and mathematical approaches to meta-heuristic and evolutionary-based algorithms such as genetic algorithms, simulated annealing, particle swarm optimization, ant colony optimization, and shuffled frog leaping. Some studies assumed that duration and cost of activities are deterministic and some other papers investigated the nondeterministic nature of the activities; some attempted to solve the problem with a single objective of minimizing either time or cost; and some other examined it as a multi-objective problem minimizing both time and cost simultaneously. However, due to the complexity of the problem, the applications of all these TCT optimization models in the literature are mainly limited to very small demonstration cases with a few numbers of activities that are far less than what exist in real-life construction projects.

To expand the available solution methods to more practical scale problems, some research studies have utilized the power of supercomputers and/or parallel computing techniques to overcome the lengthy computing time for solving large-sized problems. In these studies, calculations of the optimization process are distributed over a network of several computers and each processor/computer processes the allocated chunks of the optimization in order to reduce the overall computation time. For example, in a research study in 2005 (Kandil & El-Rayes, 2005), the optimization time of 434 hours for a network of 720 activities on a single

processor was reduced to 54 hours with the help of a parallel computing framework using 50 processors. In another similar effort using parallel genetic algorithms, the computational time for a 720 activity project was reduced from 136.5 hours (i.e., 5.7 days) of continuous computations on a single processor to 6.7 hours when executed over a cluster of ten processors (Kandil & El-Rayes, 2006). However, using supercomputing clusters or utilizing the benefits of the parallel computing frameworks is not an accessible and practical solution for most construction firms.

In the current highly competitive and unstable business environment, the ability of construction firms to optimize their projects and to monitor progress within strict cost, time, and performance guidelines is becoming increasingly important (Chen & Tsai, 2011). The purpose of this research is to utilize recent developments in computation technology and prominent optimization techniques, such as Constraint Programming (CP), to improve the current limitations in solving large-scale Discrete Time-Cost Trade-off (DTCT) problems. Constraint programming, a novel approach for solving problems with discrete decision variables (Integer or Boolean), has opened up new prospects for solving large-scale real-world optimization problems.

## 1.2 Research Motivation

This research has three main motivations: the complex nature of time-cost trade-off problems; the inefficiency of traditional optimization methods for solving large-scale TCT problems; and the potential use of advanced tools and novel techniques for circumventing the limitations of traditional optimization methods. These are briefly described as follows:

### 1.2.1 The Complex Nature of TCT Problems

In the literature, discrete time-cost trade-off problem is classified as combinatorial NP-hard (De, Dunne, Ghosh, & Wells, 1997) which is the category of problems with no efficient algorithm. The solution to this type of problems exhibit worse case complexity (De et al., 1997): when the size of the problem grows, the computation time for solving it would grow as an exponential function of the problem size (De, James Dunne, Ghosh, & Wells, 1995). As a result, solving large combinatorial problems is very time-consuming and prohibitive.

The goal in solving such type of problems typically is to find a satisfactory near optimum solution within an acceptable processing time, rather than finding the global optimum solution that may take a substantial impractical amount of time.

### 1.2.2 Inefficiency of Traditional Methods for Optimizing Large-Scale Problems

Many optimization models have been proposed to optimize the trade-off between time and cost in construction projects. Mathematical optimization methods such as linear programming, integer programming, and dynamic programming are among the primary optimization methods which were used to solve relatively small TCT problems. Linear programming is an appropriate method for solving problems with linear time-cost relationships, but fails to solve problems with discrete time-cost relationships (C.-W. Feng, Liu, & Burns, 1997). Integer programming and dynamic programming require a lot of computational effort for solving more complex project networks or for solving projects with numerous activities.

Evolutionary-based optimization methods, as alternative methods of optimization, were introduced to address the shortcoming of mathematical optimization methods for solving large TCT problems. In recent decades, various evolutionary-based optimization methods including genetic algorithms, particle swarm, simulated annealing, ant-colony, and leap frog optimization have been applied for solving TCT problems. Although these alternative optimization methods have some advantages over the mathematical optimization methods, and have been applied with success for optimization of many TCT problems, they are still not efficient in optimizing large-scale project networks. In the literature, the application of these methods for solving TCT problems is limited to projects comprising limited number of activities, mostly less than one hundred activities (Kandil & El-Rayes, 2005); for large-scale projects they require impractical processing times to find a near optimal solution.

### 1.2.3 Potential Use of Advanced Tools and Techniques

With recent improvements in computing technologies and optimization algorithms, more complicated problems can be solved today than ever before (P. Van Hentenryck & Michel, 2009). Recent optimization techniques such as Constraint Programming has opened up new prospects for optimization of large-scale problems; and big software companies such as IBM, Microsoft, and Google (Operations Research Tools developed at Google) have either acquired emerging optimization companies or have started their investments in this field.

Constraint programming is a novel approach for solving computationally hard problems with discrete variables. It is a hybrid technology based on various optimization techniques which is expected to solve large-scale combinatorial optimization problems. A key motivation for this research is to utilize advanced optimization tools and techniques such as constraint

programming to improve the existing limitations of traditional optimization methods on optimizing time and cost in large construction projects.

## 1.3 Research Objectives

The objective of this research is to examine the performance of traditional optimization techniques including mathematical and evolutionary-based methods on solving large-scale TCT problems. Furthermore, the goal is to improve the solutions by utilizing state-of-the-art optimization tools and techniques to solve real-life large-scale TCT problems. The detailed research objectives are as follows:

1. Investigate the performance of mathematical optimization methods by utilizing advanced mathematical optimization tools for optimizing TCT problems. This comprises choosing the most appropriate optimization tools, developing an optimization model and performing several experimentations on the model with various numbers of activities.

2. Examine the performance of evolutionary-based optimization methods by utilizing advanced evolutionary-based and genetic algorithms optimization tools. This includes several experimentations for solving TCT problems with various numbers of activities using a model similar to the previously mentioned model.

3. Validate the potential of new emerging optimization tools and techniques such as constraint programming for solving large-scale TCT problems. The validation consists of modeling and optimizing TCT problems of several sizes and comparing

and discussing the CP optimization results with the mathematical and evolutionary-based optimization results.

4. Recommend the best available tools and techniques for optimization of large-scale construction projects to obtain the best near optimal results within a reasonable amount of time and computational effort.

This research supports the efforts of construction firms for optimizing time and cost of their real-world construction projects, particularly because none of the commercial scheduling and project management software packages such as Microsoft Project and Primavera include this vital feature.

## 1.4 Research Methodology

The methodology to achieve the research objectives is demonstrated in the following steps:

1. **Extensive Literature Review:** An extensive literature review of time-cost trade-off analysis and optimization methods.
2. **Investigation of Optimization Tools:** An investigation of available optimization tools and techniques for solving large-scale optimization problems to decide on the most appropriate ones to be used for the purpose of this research.
3. **Development of Optimization Models:** Developing the required TCT optimization models to be used for various experimentations.
4. **TCT Optimization using Mathematical Methods:** Optimizing project networks with various sizes using mathematical optimization methods.

5. **TCT Optimization using Evolutionary-Based Methods:** Optimizing project networks with various sizes using evolutionary-based optimization methods.
6. **Constraint Programming Optimization:** Modeling and performing experimentations for TCT optimization with various project sizes using constraint programming methods.
7. **Validation:** validate the superiority of constraint programming methods for solving large-scale TCT problems by analyzing and comparing the results of mathematical, evolutionary-based and constraint programming optimization methods.
8. **Conclusion and Discussions:** conclude on the best available tools and techniques for optimization of large-scale construction projects and the value of this research to other contemporary optimization problems in some other fields.

## 1.5 Thesis Organization

The reminder of the thesis is organized as follows:

**Chapter 2** presents the literature review in three distinct sections. The first section is dedicated to the time-cost trade-off (TCT) analysis. It includes the definition and history of TCT problem. It also discusses solution challenges, various categories, and a simple mathematical formulation of the TCT problem.

The second section describes the current and proposed TCT optimization methods. It includes the formal definition and brief explanations about mathematical, evolutionary-based and constraint programming optimization methods.

The third section provides an overview of existing tools for solving large-scale optimization problems and draws attention to the differences between available tools. This chapter concludes on the proper tools for this research.

**Chapter 3** focuses on the comparison of traditional optimization methods for solving TCT problems including mathematical and evolutionary-based optimization methods. This chapter starts with describing the details of a TCT model for the chapter's experimentations and continues with the actual experiments using mathematical and evolutionary-based optimization methods. The results of this chapter would be a basis for chapter 4, where the results of mathematical, evolutionary-based and constraint programming techniques are compared and discussed.

**Chapter 4** concentrates on the TCT optimization using constraint programming techniques. This chapter describes in more detail the constraint programming optimization method, its similarities and differences to mathematical optimization methods, and introduces the IBM ILOG Optimization Studio as the leading commercial package for using constraint programming techniques. An optimization model is then developed using Optimization Programming Language (OPL) and various experiments are performed using IBM ILOG Optimization Studio.

**Chapter 5** discusses the results of traditional optimization methods and constraint programming method and concludes on the best available tools and techniques for optimization of project networks. It also outlines proposed research studies that have the potential to be accomplished based on the work of this research.

# Chapter 2
# Literature Review

## 2.1 Introduction

This chapter presents a review of the time-cost trade-off (TCT) problem, explores different categories of the problem in the literature, and investigates the optimization methods for solving the TCT problem. The investigation includes a detailed description of existing traditional optimization techniques such as mathematical programming and evolutionary-based optimization methods, and also introduces constraint programming as a prominent non-traditional optimization technique for solving TCT problems. Available optimization tools for solving TCT optimization problems are then summarized, the differences are discussed, and the most appropriate ones are chosen for the purpose of various experimentations for this research.

## 2.2 Time-Cost Trade-off Analysis

The Critical Path Method (CPM) is a useful scheduling technique only when the project deadline is not fixed (Liao, Egbelu, Sarker, & Leu, 2011). To use CPM for a project with a fixed deadline or for a project which is running behind schedule, the TCT analysis is implemented to meet the project deadline. In the TCT analysis some of the activities on the critical path are substituted with their shorter modes of construction to save time. In addition, non-critical activities are relaxed to save cost (Siemens, 1971). The results of this analysis are a time-cost trade-off curve (e.g., Figure 2-1) showing the relationship between project duration and cost under different decisions and the selection of construction methods that provide the optimal balance of project duration and cost (T. Hegazy, 1999).

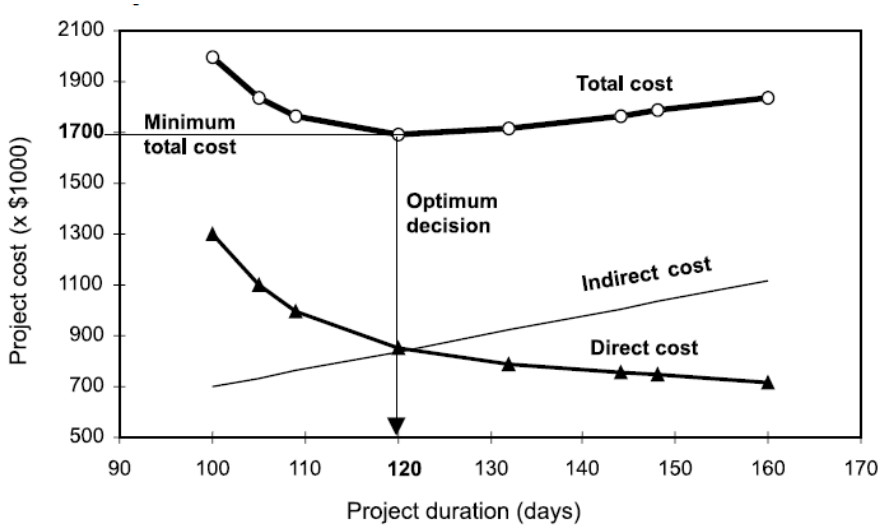


**Figure 2-1 Project Time-Cost Relationship (Tarek Hegazy, 2001)**

The TCT analysis involves optimal selection of some activities on the critical path of the project to use their faster and more expensive modes of construction. It is, therefore, unavoidable that the cost of these compressed activities will be increased as a result. The "normal time" for completing an activity is determined by calculating the minimum cost ("normal cost") for the activity. The minimum duration for an activity is known as the "crash time", and the cost associated with the crash time is called "crash cost". Any intermediate point in between the normal and crash points can be computed in accordance with the activity time-cost relationship (Liao et al., 2011).

The activity time-cost (direct cost) relationship can be linear or curvilinear, continuous or discrete. Figure 2-2 presents different relationships between activity time and direct cost.

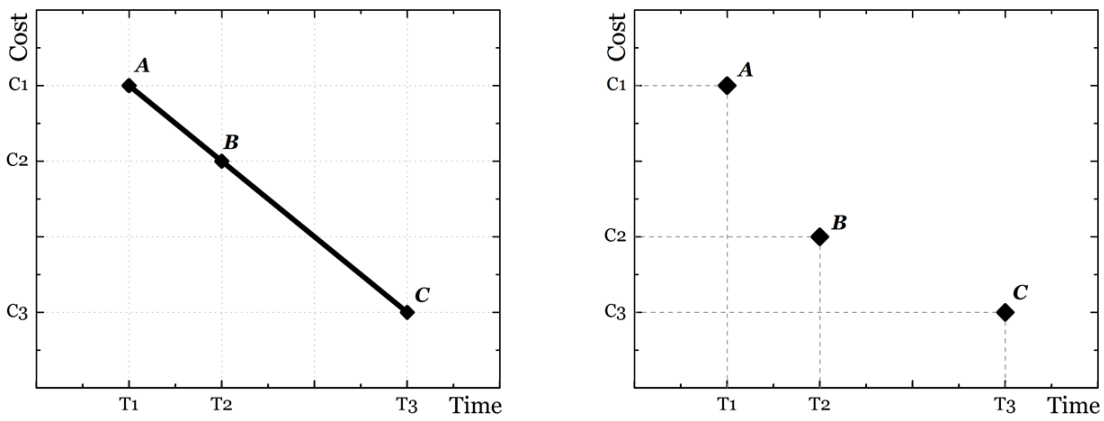


**Figure 2-2 Linear and Discrete Relationships of Activity Modes**

The early attempts to incorporate time-cost trade-offs in project networks assumed that the activity costs are a linear function of the activity durations, which are bounded from below to normal duration and from above to crash duration. The objective was to determine the activity durations and to schedule the activities in order to minimize the project costs (Vanhoucke & Debels, 2007). Several other forms of activity cost functions have been studied over the years such as concave, convex and discrete.

Since the discrete relationship between activity time and cost is the most representative version for real-world projects, the discrete time-cost trade-off problem (DTCTP) is the most practical version. It involves the selection of a set of execution modes for each activity in order to optimize time or cost, or both. The focus of this research is on the discrete version of the time-cost trade-off problem.

## 2.3 TCT Optimization Challenges

Optimization is the process of trying to find the best solution to a problem that may have many possible solutions (*The Genetic Algorithm Solver for Microsoft Excel*, 2010). Once the

search space of the problem becomes too large for the calculating power of available computers, finding the optimal solution among all other feasible solutions to the problem may take a substantial and an impractical amount of time.

With real-life projects involving hundreds or thousands of activities, finding optimal TCT modes of construction is difficult and time consuming considering the number of permutations involved (Liang, Burns, & Chung-Wei, 1995). Evaluating each alternative requires recalculation of the schedule using the critical path method (CPM) and reassessment of total project cost. Exhaustive enumeration is, therefore, not a feasible and practical solution even with very fast computers (T. Hegazy, 1999).

In fact, available construction optimization models can be used to generate optimal trade-off between construction time and cost; however, their applications in optimizing large-scale projects are limited due to their extensive and impractical computational time requirements (Kandil & El-Rayes, 2005).

## 2.4 Different Categories of TCT in the Literature

The time-cost trade-off problems have been extensively investigated during the last decades. Many researchers, therefore, have explored different aspects of the problem. Studied aspects of the problem can broadly be classified into three categories: 1) deterministic or uncertain characteristics of the activities; 2) single or multi-objective essence of the objective function; and 3) the solution methods which have been used to solve the problem.

1) In traditional time-cost trade-off analysis, the time and cost of activities are assumed to be deterministic (Chung-Wei Feng, Liu, & Burns, 2000). However, activity time and cost are realistically uncertain (Liao et al., 2011). A general overview of this category is done by (Chen & Tsai, 2011). For the deterministic case, typical articles include (Kelley, 1961; Siemens, 1971; Phillips & Dessouky, 1977; Talbot, 1982; Liang et al., 1995; T. Hegazy, 1999). On the other hand, several stochastic models have been developed to address time-cost trade-off problems with uncertain activity durations, such as (Charnes, Cooper, & Thompson, 1964; Golenko-Ginzburg & Gonik, 1997; Gutjahr, Strauss, & Wagner, 2000; Chung-Wei Feng et al., 2000; Ke, Ma, & Ni, 2009; Mon, Cheng, & Lu, 1995). Uncertain activity time and cost can be treated statistically when there is enough data available; otherwise, probabilistic models or fuzzy models are more appropriate to handle uncertainty (Liao et al., 2011).

2) Minimizing cost or time has been the most common objective of the time-cost trade-off problems in the early studies of the TCT analysis. However, time-cost trade-off problem may be treated as a multi-objective optimization process to minimize both duration and cost (Liao et al., 2011). Many studies have implemented a multi-objective approach to solve TCT problem such as C.-W. Feng et al. (1997); Chung-Wei Feng et al. (2000); Zheng, Ng, & Kumaraswamy (2004); Azaron, Perkgoz, & Sakawa (2005); Zheng, Ng, & Kumaraswamy (2005); Kandil & El-Rayes (2006); Ng & Zhang (2008); Xiong & Kuang (2008); Afshar, Ziaraty, Kaveh, & Sharifi (2009); and Castro-Lacouture, Süer, Gonzalez-Joaqui, & Yates (2009).

3) Another very important classification of TCT problems is based on the solution method being used to solve them. Techniques for modeling and solving time-cost trade-off problems can be broadly classified into three subcategories of heuristic methods, mathematical approaches, and evolutionary-based optimization algorithms (EOAs) (Ng & Zhang, 2008). The purpose of this research is to apply proper solution techniques in order to solve large-scale TCT problems. Thus, the available solution methods are described in more detail in the next section.

## 2.5 Techniques for Solving TCT Problems

Heuristic methods, mathematical programming approaches and evolutionary-based techniques are considered the traditional methods of solving TCT problems.

### 2.5.1 Heuristic Methods

Heuristic algorithms are not considered to be in the category of optimization methods. They are algorithms based on rules of thumb to find an acceptable near optimum solution. Heuristic methods are usually easy to understand algorithms which can be applied to larger problems and typically provide acceptable solutions (T. Hegazy, 1999). However, they lack mathematical consistency and accuracy and are specific to certain instances of the problem. Fondahl, 1962; Prager, 1963; Siemens, 1971; and Moselhi, 1993 are some of the research studies that have utilized heuristic methods for solving TCT problems.

#### 2.5.1.1 A Sample TCT Heuristic Method

The cost-slope method is a simple heuristic approach for solving TCT problems. This method shortens the project duration assuming that the relationship between time and cost is linear.

According to this assumption, the cost slope of an activity is defined as the rate at which the direct cost increases when its duration is shortened by a unit of time (Tarek Hegazy, 2001).

The detailed steps of the cost-slope method are as follows:

1. Use normal durations and costs for all activities.

2. Calculate the CPM and identify the critical path.

3. Eliminate all non-critical activities.

4. Tabulate normal/crash durations and costs for all critical activities.

5. Compute and tabulate the "cost-slope" of each critical activity:

$$Cost\ Slope = \frac{Crash\ Cost - Normal\ Cost}{Normal\ Duration - Crash\ Duration}$$

6. Identify the critical activity with the least cost slope and possible duration shortening.

7. Reduce the duration of this activity until its crash duration is reached or until the critical path changes.

8. If the network has more than one critical path, we need to shorten both of them simultaneously. This can be done by shortening a single activity that lies on all paths or by shortening one activity from each path. The option to choose is determined by

comparing the cost slope of the single activity versus the sum of cost slopes for the individual activities on all critical paths.

9. Calculate the direct cost increase due to activity shortening by multiplying the cost slope by the time units shortened. Add the additional cost to the total direct cost.

10. If float times are introduced into any activity, relax them to reduce cost.

11. Plot one point (project duration versus total direct cost) on a figure such Figure 2-1.

12. Continue from Step 2 until no further shortening is possible to the project.

13. Plot indirect project costs on the same figure. Add the direct cost + indirect cost and plot the total cost curve.

14. Get the optimum TCT strategy as the one with minimum total cost.

An example of a complete case study solved based on the cost-slope heuristic method can be found in (Tarek Hegazy, 2001). Since heuristic algorithms are not in the category of optimization methods, in this research their application for solving TCT problems is limited to introducing the cost-slope method.

### 2.5.2 Mathematical Programming

Mathematical programming or mathematical optimization is the category of optimization methods in which mathematical methods such as linear programming, integer programming,

dynamic programming, and nonlinear programming are implemented to solve an optimization problem.

The simplex method, developed in 1947 by George B. Dantzig, demonstrated extraordinary computational efficiency and robustness for solving the linear programming problems. The exceptional power of the simplex method, together with the availability of high-speed digital computers, made linear programming the most fundamental method and the starting point of mathematical optimization. Since then, many additional techniques have been developed, which relax the assumptions of the linear programming and broaden the applications of the mathematical programming approach (Bradley, Hax, & Magnanti, 1977).

Mathematical programming methods are the primary optimization methods implemented to solve time-cost trade-off problems. Some of the papers that have applied mathematical optimization methods for solving TCT problems include Kelley (1961); Meyer & Shaffer (1963); Patterson & Huber (1974); Robinson (1975); Hendrickson & Au (1989); Pagnoni (1990); Elmaghraby (1993); De, James Dunne, Ghosh, & Wells (1995); and Burns, Liu, & Feng (1996).

#### 2.5.2.1 Basic Time-Cost Trade-off Formulation

The common mathematical formulation for time-cost trade-off relies on the linear relationship between time and cost. This relationship for an activity *i* is shown in Figure 2-3 in addition to the common terms used in the formulation.

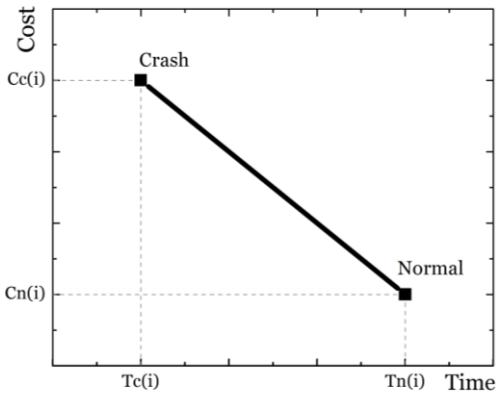


**Figure 2-3 Linear Relationship of Time and Cost for Activity i**

$C_n(i) = Normal\ cost\ for\ activity\ i$

$C_c(i) = Crash\ cost\ for\ activity\ i$

$T_n(i) = Normal\ time\ for\ activity\ i$

$T_c(i) = Crash\ time\ for\ activity\ i$

$T(i) = Intermidiate\ time\ for\ activity\ i$

$C(i) = Intermidiate\ cost\ for\ activity\ i$

The "cost slope" ratio for the activities on the critical path is then the main principle for selecting the activities to be crashed in order to reduce the total project duration. The cost slope $a_i$ of an activity $i$ is the rate at which the direct cost increases when its duration is shortened by a unit of time (Tarek Hegazy, 2001). The formulation for a single objective time-cost trade-off is (Li & Love, 1997)

$$Min\ C_t = \sum_{\forall i} C_i = \sum_{\forall i} [-a_i\, T(i) + b_i]$$

Subject to

$$\sum_{\forall i} T(i) = T_t$$

$$a_i = [C_c(i) - C_n(i)]/[T_n(i) - T_c(i)]$$

$$b_i = [C_c(i)T_n(i) - C_n(i)T_c(i)]/[T_n(i) - T_c(i)]$$

$$T_c(i) \le T(i) \le T_n(i)$$

where $i$ is any activity on the critical path which can be crashed, $C_t$ is the total cost and $T_t$ is the required total crash time for the project.

As discussed in Section 2.2, the linear relationship between time and cost for an activity is not usually a practical assumption, as activities cannot be crashed in any fraction of time. In fact in most construction projects, the minimum time fraction is normally either half a day or a day.

### 2.5.3 Evolutionary-Based Optimization Algorithms

The difficulties associated with using mathematical methods for solving large-scale optimization problems have contributed to the development of alternative solutions. Linear programming and dynamic programming techniques, for example, often fail (or reach local optimum) in solving NP-hard problems with large number of variables and non-linear objective functions. To overcome these problems, researchers have proposed evolutionary-based algorithms for searching near optimum solutions to problems (Elbeltagi, Hegazy, & Grierson, 2005).

Evolutionary-based Optimization Algorithms (EOAs) are stochastic search methods that mimic the natural biological evolution of species and/or their social behavior. These algorithms have been developed to solve large-scale optimization problems, for which traditional mathematical techniques may fail (Elbeltagi et al., 2005). Various research studies have discussed evolutionary-based optimization methods for solving TCT problems including C.-W. Feng, Liu, & Burns (1997); Elbeltagi et al. (2005); Azaron, Perkgoz, & Sakawa (2005); Zheng, Ng, & Kumaraswamy (2004), (2005); Kandil & El-Rayes (2006); Xiong & Kuang (2008); Ng & Zhang (2008); and Afshar, Ziaraty, Kaveh, & Sharifi (2009).

Various evolutionary-based algorithms such as genetic algorithms, particle swarm, simulated annealing, ant-colony, and leap frog shuffling have been used for optimizing TCT problems as well as optimizing various problems in many other fields; however, Genetic Algorithms optimization methods are the most widely used EOA in the literature.

#### 2.5.3.1 Genetic Algorithms

Genetic algorithms (GAs), invented by John Holland in in the 1960s and the 1970s at the University of Michigan, are the most widely used evolutionary-based computation approach. Genetic algorithms (GAs) are search methods based on genetics and evolution principles and are mainly useful for highly non-linear problems and models for which the computation time is not a primary concern. Although numerous revised algorithms are available, a GA typically proceeds in the following order (Schreyer, 2011):

1) Start with an initial finite population of randomly chosen chromosomes (Parent population) in the design space. This population constitutes the first generation (iteration).

2) Evaluate each member of the population with their fitness function value.

3) Rank the chromosomes by their fitness.

4) Apply genetic operators (mating): reproduction (reproduce chromosomes with a high fitness), cross-over (swap parts of two chromosomes, chosen based on their fitness to create their offspring) and mutation (apply a random perturbation to parts of a chromosome). All of these operators are assigned a probability of occurrence.

5) Evaluate the fitness of the new generation from their chromosomes.

6) Continue genetic mating as specified in step 4 and iterate until convergence is achieved or the process is stopped.

Holland's original genetic algorithm was quite simple, yet remarkably robust, and could find optimal solutions to a wide range of problems. Many recent genetic algorithm programs that solve very large and complex real-world problems use only slightly modified versions of the original genetic algorithm (*The Genetic Algorithm Solver for Microsoft Excel*, 2010).

Due to the improved results, genetic algorithms have been successfully used to solve several engineering and construction management problems. In the literature, Genetic algorithms have been the dominant methods of solving TCT problems during the last decade.

### 2.5.4 Constraint Programming

Constraint programming is a novel approach for solving large-scale optimization problems with discrete variables. It is a hybrid method to effectively reduce the domain of decision variables and search for the best feasible solution in the reduced domains. This new optimization technique will be described in more detail in Chapter 4, Section 4.1 – An Introduction to Constraint Programming.

### 2.5.5 Summary of Solution Techniques

Mathematical programming methods including linear programming, integer programming, and dynamic programming can provide optimal solution to the problem; however, their formulation for discrete TCT problem is complex and usually fails to solve large problems.

Evolutionary-based optimization methods are search methods which mimic either the evolution process of species or the natural behaviors of creatures to find near optimum solution to problems. These methods can be applied to larger problems; however, the processing time to reach to an acceptable near optimum solution may be too large. Genetic Algorithm is one of the most common evolutionary-based algorithms for solving TCT problems. Table 2-1 represents a comparison of the common traditional methods for solving TCT problems and Figure 2-4 provides an outline of TCT analysis methods.

**Table 2-1 Traditional Techniques for Time-Cost Trade-off Analysis (based on T. Hegazy, 1999)**

| | Heuristic Methods | Mathematical Programming | Evolutionary-Based Algorithms |
|---|---|---|---|
| **Description** | *• Simple rule of thumb (e.g. Crashing the cheapest critical activities)* | *• Linear programming • integer programming • dynamic programming* | *• Optimization search procedures that mimic natural evolution or social behavior of species* |
| **Advantages** | *• Easy to understand • Provide good solutions • Used for large projects* | *• May provide optimal solution* | *• Robust search algorithm • Can use discrete relationship between time and cost • Applicable to large problems* |
| **Drawbacks** | *• Lack mathematical rigor • Do not guarantee optimal solution • Mostly assume linear, rather than discrete, relationship between time and cost* | *• Difficult to formulate • May terminate in a local optimum • Applies to small problems only • Mostly assume linear, rather than discrete relationship between time and cost* | *• Random search is time consuming • Cannot dertermine when or if an optimal solution is reached* |

Constraint programming is a novel approach for solving large-scale optimization problems with discrete variables (Integer or Boolean) which will be discussed in more detail in Chapter 4.

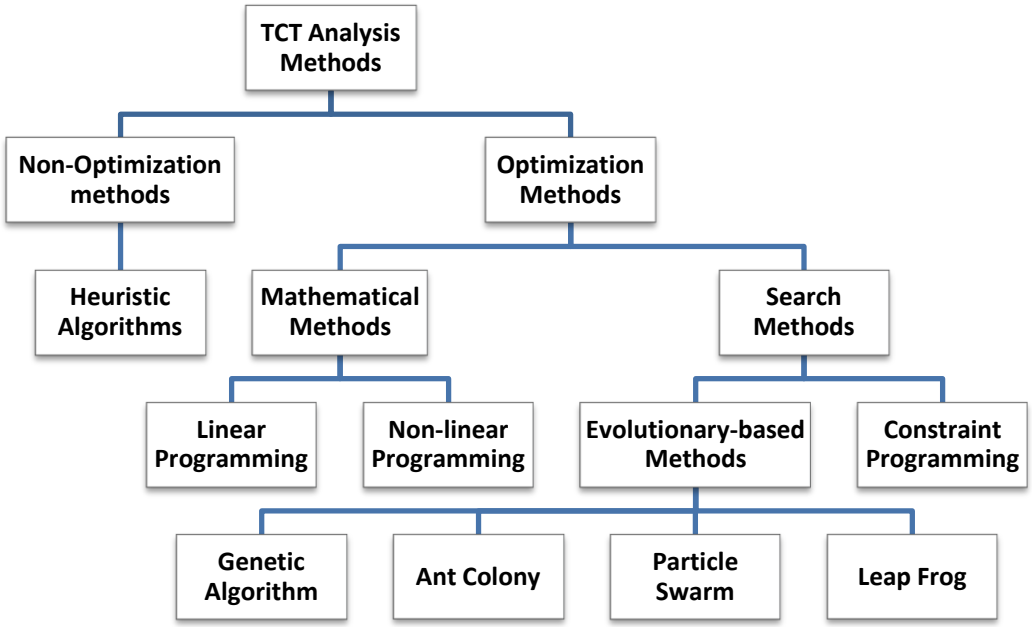


**Figure 2-4 Overview of TCT Analysis Methods**

Despite considerable research in the literature for solving discrete time-cost trade-off problem using various solution techniques, little effort has been made in extending the problem to more realistic settings to solve large-scale problems (Vanhoucke & Debels, 2007). In the next section available tools for solving large-scale optimization problems are discussed to conclude on the best ones for optimization purposes of this research.

## 2.6 Tools for Modeling and Solving Large-Scale Optimization Problems

The solution to any optimization problem requires two separate stages: Modeling and Solving as shown in Figure 2-5. Both of these stages are very important to achieve decent results.

Various commercial optimization tools are available to effectively model and efficiently solve different types of optimization problems. Some of these tools are exclusively designed for modeling purposes; certain tools are merely solvers (or optimization engines); and particular ones are optimization packages, comprising both modeling, as well as, solving capabilities.

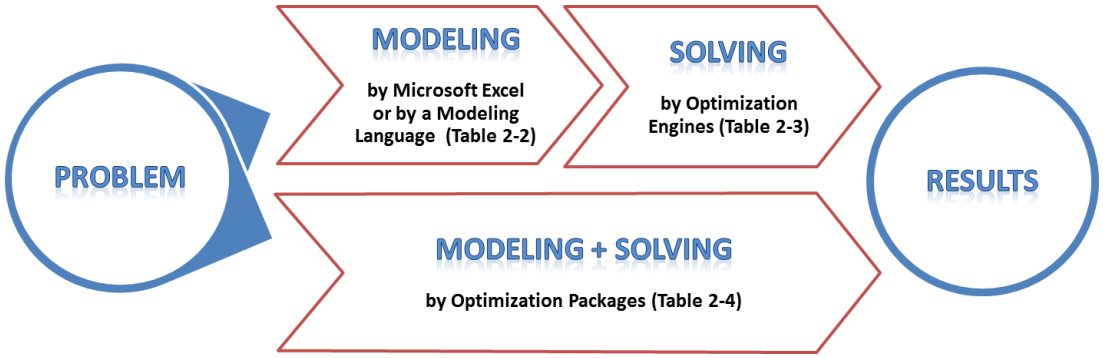


**Figure 2-5 Solving an Optimization Problem**

Typically, optimization models have four main components: 1) an objective function, expressing the main objective of the model, to be either minimized or maximized, 2) a set of decision variables, which control the value of the objective function, 3) a set of constraints, which control the variables to take on certain values but exclude others, and 4) data sets, which are the known coefficient of decision variables. The entire model builds a relationship between the objective function, constraints, decision variables, and known data. The optimization problem is then to find values of the variables that minimize or maximize the objective function while satisfying the constraints.

Modeling of an optimization problem is commonly performed by either of the following alternatives: 1) modeling languages, 2) programming languages, or 4) Microsoft Excel. Each of these modeling options has their own strength; however, utilized modeling tool should be

well-matched with the solver which is going to solve the model. In the following sections of this chapter, first, advantages and drawbacks of each modeling alternative are discussed; then, common tools for modeling and solving optimization problems are presented; and finally, the most appropriate optimization tools for the purpose of this research are carefully selected.

### 2.6.1 Modeling Languages vs. Programming Languages

Developing an optimization model can be accomplished using a modeling language or using a general-purpose programming language such as C, C++ or Java. However, implementing a model in a modeling language is often much more efficient, and the final model is significantly more compact with less lines of code. Moreover, optimization languages eliminate the difficulties associated with low level programming issues such as memory allocations and pointers. Tweaking the model to achieve the best performance is also more convenient using optimization languages. Optimization languages yield improvements in efficiency and accelerate the development of optimization models which can in turn be valuable for spending more time to improve the model formulation (Kalvelagen, 2009).

Common modeling languages are listed and the advantages are explained in more detail in Section 2.6.3.

### 2.6.2 Microsoft Excel vs. Modeling Languages

Modeling in Microsoft Excel has many clear advantages. Excel is widely available and has many users who are familiar with its structure and usage. In Excel, cells and cell-references are used to formulate and implement optimization models. The direct implementation of a model in Excel has also numerous additional benefits such as availability of data manipulation

tools and report writing facilities including the availability of numerous built-in functions, dynamic graphs, and pivot tables. There are also some disadvantages: spreadsheet modeling is prone to errors, and the model lacks structure and dimensionality (Kalvelagen, 2009).

Alternatively, using modeling languages to build optimization models requires a steep learning curve to master the modeling details and techniques. However, once mastered, they are quite efficient for modeling complicated problems. In addition, most of the modeling languages provide an interface to access a wide range of state-of-the-art solvers which can be used to solve the model.

### 2.6.3 Modeling Languages

Optimization modeling languages, often referred to as "Algebraic Modeling Languages", are designed for describing and solving large-scale complex problems in a concise and readable format. These languages are high-level declarative programming languages that specify what is being computed rather than how it is done (Gassmann & Ireland, 1995). The syntax of modeling languages closely resembles the notation of optimization problems which is a clear advantage to simplify the modeling process. Table 2-2 represents some of the common optimization modeling languages.

AMPL, GAMS, and OPL are among the most common modeling languages. A Mathematical Programming Language (AMPL) designed in 1985 at Bell Laboratories and is still one of the most popular modeling languages for large-scale linear, mixed integer and nonlinear optimization problems. It is a modeling language for formulating mathematical problems; however, many solvers can be used to solve the model implemented by AMPL.

The General Algebraic Modeling System (GAMS) was the first algebraic modeling language. It is specifically designed for modeling linear, nonlinear and mixed integer mathematical optimization problems. GAMS is a modeling language which contains an integrated development environment (IDE) and is connected to a group of third-party optimization solvers.

**Table 2-2 Common Algebraic Modeling Languages**

| Software | Developer | Description |
|---|---|---|
| AMPL | AMPL Optimization LLC | An algebraic modeling language for LP and NLP optimization problems |
| General Algebraic Modeling System (GAMS) | GAMS Development Corporation | A high-level modeling system for LP, NLP, and MIP |
| Optimization Programming Language (OPL) | IBM Corporation | An optimization modeling system for mathematical programming and constraint programming |
| AIMMS | Paragon Decision Technology Inc. | An advanced modeling system |
| Optimization Modeling Language (OML) | Microsoft Corporation | An algebraic modeling language designed exclusively for modeling within Microsoft Solver Foundation |
| Xpress-Mosel | FICO | A language that is both a modeling and a programming language |

Optimization Programming Language (OPL) is a modeling language designed specifically for optimizing combinatorial problems. The design of OPL was motivated by the successful implementation of mathematical modeling languages like AMPL and GAMS. Like AMPL and GAMS, OPL provides full support for linear programming and integer programming. The

main difference between OPL and other algebraic modeling languages is that OPL also provides support for constraint programming.

### 2.6.4 Optimization Engines (Solvers)

Modeling languages define the optimization problems in an appropriate format to be solved by optimization engines. It is the responsibility of the optimization engines to analyze the model, and if no error found, solve the model and provide the solutions to the problem. Table 2-3 presents a list of common optimization engines.

**Table 2-3 Common Optimization Engines**

| Software | Developer | Description |
|---|---|---|
| CPLEX Optimizer | IBM Corporation | A solver for linear and nonlinear optimization problems |
| CPLEX CP Optimizer | IBM Corporation | A solver for optimization based on constraint programming technique |
| Gurobi | Gurobi Optimization | A state-of-the-art solver for LP, QP, MILP and MIQP |
| MOSEK | MOSEK ApS | A solver for LP, QP, Conic QP, and Convex NLP programming |
| Xpress-Optimizer | FICO | A solver for large-scale LP, MIP, QP, and MIQP problems |
| KINTRO | Ziena Optimization, Inc. | A solver for LP, QP, and nonlinear optimization |

IBM ILOG CPLEX Optimizer and Gurobi Optimizer are among the most popular optimization engines. CPLEX Optimizer provides flexible, high-performance mathematical

programming for linear programming, mixed integer programming, quadratic programming, and quadratically constrained programming problems. The CPLEX Optimizer is accessible through many modeling languages including AMPL, GAMS, and OPL.

The Gurobi Optimizer is a state-of-the-art solver for linear programming (LP), quadratic programming (QP) and mixed-integer programming (MILP and MIQP). The Gurobi Optimizer is also accessible through many modeling languages including AMPL and GAMS.

IBM ILOG CPLEX CP Optimizer is the constraint programming counterpart of CPLEX Optimizer. CP Optimizer uses constraint programming technology to solve detailed scheduling problems and other hard combinatorial optimization problems.

### 2.6.5 Optimization Packages

Many commercial optimization packages facilitate modeling of optimization problems, as well as, solving the models through a set of integrated optimization engines all in a single package. Particular optimization packages, such as, Frontline Solvers and Evolver are Excel-based in which the model is developed in Microsoft Excel and the solution to the optimization problem is also presented in Excel. Moreover, several optimization packages such as IBM ILOG Optimization Studio, Microsoft Solver Foundation, and Xpress Optimization Suite provide a proprietary optimization modeling language in addition to one or more exclusive optimization engines. Typically, they also offer particular interfaces in order to provide connection to external modeling languages and solvers. Some of the common optimization packages are presented in Table 2-4.

Frontline Solvers, including Premium Solver Platform and Risk Solver Platform, are among the best Excel-based optimization packages that offer the flexibility of modeling with Excel and the power of connecting to a variety of commercial optimization engines for solving the problems. Risk Solver Platform and Premium Solver Platform have a limit of 8,000 decision variables for linear problems, and 1000 variables for nonlinear problems (*Frontline Solvers User Guide*, 2011).

**Table 2-4 Common Optimization Packages**

| Software | Developer | Description |
|---|---|---|
| Risk Solver Platform | Frontline Systems Inc. | Risk analysis, simulation, and optimization tools |
| Evolver | Palisade Corporation | GA-based optimization tool effective in optimizing complex and large-scale models |
| IBM ILOG CPLEX Optimization Studio | IBM Corporation | A comprehensive platform for mathematical and constraint programming optimization |
| Microsoft Solver Foundation (MSF) | Microsoft Corporation | A set of development tools for mathematical simulation, optimization, and modeling |
| Excel Solver | Microsoft Corporation | Easy-to-use mathematical optimization tool capable of handling simple LP and NLP problems |
| What's Best! | LINDO Systems | An Excel Add-In for LP, NLP, and MIP Modeling and Optimization |
| LINGO | LINDO Systems Inc. | An Optimization Software for LP, NLP, and IP |
| Xpress Optimization Suite | FICO | A modeling and optimizatoin package for developing and solving large scale models |

Evolver is a genetic algorithm optimization add-in for Microsoft Excel. Evolver uses innovative genetic algorithm (GA) technology to solve optimization problems. Many types of problems including complex nonlinear problems can be solved by Evolver. Evolver's genetic algorithms has proven to be one the most efficient GA algorithms in the market and can lead to the best overall global solution.

The IBM ILOG CPLEX Optimization Studio is one of the leading mathematical and constraint programming optimization packages. It consists of the CPLEX Optimizer for mathematical programming, the IBM ILOG CPLEX CP Optimizer for constraint programming, the Optimization Programming Language (OPL), and a tightly integrated IDE.

Microsoft Solver foundation is the Microsoft optimization package for solving mathematical and constraint programming problems. It includes Optimization Modeling Language (OML) for modeling, some built-in solvers and the option to be integrated to popular external solvers.

Excel Solver is the easy-to-use mathematical optimization tool packed as an Add-in in Microsoft Excel. However, the default version of Excel Solver is only capable of solving optimization problems consisting of up to 200 decision variables and 500 constraints.

### 2.6.6 Optimization Tools Used in This Research

In this research, in order to compare the results of mathematical programming versus evolutionary-based optimization methods, three Excel-based optimization packages are used: Excel Solver for mathematical optimization, Risk Solver Platform for mathematical as well as

evolutionary-based optimization, and Evolver for genetic algorithms optimization. To implement constraint programming methods, however, no Excel-based tool is available. IBM ILOG Optimization Studio and OPL language are used in this research to model and solve TCT optimization problems using constraint programming techniques.

## 2.7 Conclusions

In this chapter, the literature review was discussed in three distinct sections. The first section was dedicated to the time-cost trade-off (TCT) analysis. The definition and history of the problem, solution challenges, and various categories of the problem were discussed in this section.

The second section described the current and proposed TCT optimization methods. It included the formal definition and brief descriptions of heuristic, mathematical, and evolutionary-based methods. It also included a sample heuristic method, a simple mathematical formulation of the TCT problem, and an overview of genetic algorithms, the most well-known evolutionary-based optimization method.

The third section provided an overview of existing tools for solving large-scale optimization problems and discussed the differences and similarities between available tools. This section concludes on the optimization tools selected to be used in this research.

In the next chapters these tools and techniques will be used in several experimentations to evaluate and validate their potential for solving large-scale TCT problems.

# Chapter 3
# Comparison among Traditional TCT Optimization Methods

## 3.1 Introduction

In this chapter, a time-cost trade-off model is developed based on a project example consisting of several construction activities where each activity has three possible modes of construction. This TCT model is the framework for various experimentations in order to investigate the efficiency of mathematical and evolutionary-based optimization methods for solving large-scale TCT problems.

The investigation starts first by applying mathematical optimization methods, and then by applying evolutionary-based optimization methods, to a small-size project network of 18 activities, and continues by gradually increasing the project size to 36, 54, 180, 360 activities (and more if necessary). In each of these experiments, if a solution cannot be found in a certain predefined amount of time, the calculation is stopped and the best result is recorded. The optimization results of using mathematical and evolutionary-based methods are then compared and discussed.

## 3.2 The Proposed Model

The proposed TCT model for implementing mathematical and evolutionary-based optimization methods is developed in Microsoft Excel. The number of Excel-based advanced optimization tools has increased considerably in recent years and implementing the model in Excel has a significant advantage: enabling the same model to be used by different optimization tools.

The initial model consists of a project of 18 activities, each with three modes of construction, and each mode of construction has a defined time and cost. The model is, then, expanded to larger projects consisting of 36, 54, 180, 360, and more activities if necessary to facilitate experimentations on larger projects. Creating larger projects based on a smaller project of 18 activities has the following advantages:

- The optimal solution of large models can be calculated based on the optimal solution of the small model without any optimization. For very large models where the optimization process may not reach to the optimal solution, the calculated optimal solution can be used to verify the deviation of the result of the optimization process from the optimal solution. Thus, the calculated optimal solution is used as a measure of how far the optimization result is from the optimal solution and how well the optimization method performs.

- The project network of 18 activities is the project used in several previous research studies. It was originally introduced by (C.-W. Feng et al., 1997) and has also been used in a number of other research studies including T. Hegazy (1999); T. Hegazy & Ayed (1999); El-Rayes & Kandil (2005); Kandil & El-Rayes (2005); Ng & Zhang (2008); and Afshar et al. (2009). As such, it would be an acceptable framework to compare the results of this research with the previous research studies.

The activity relationships for the model project consisting of 18 activities are shown in Figure 3-1 and three modes of construction for each activity and their associated time and cost are presented in Table 3-1.

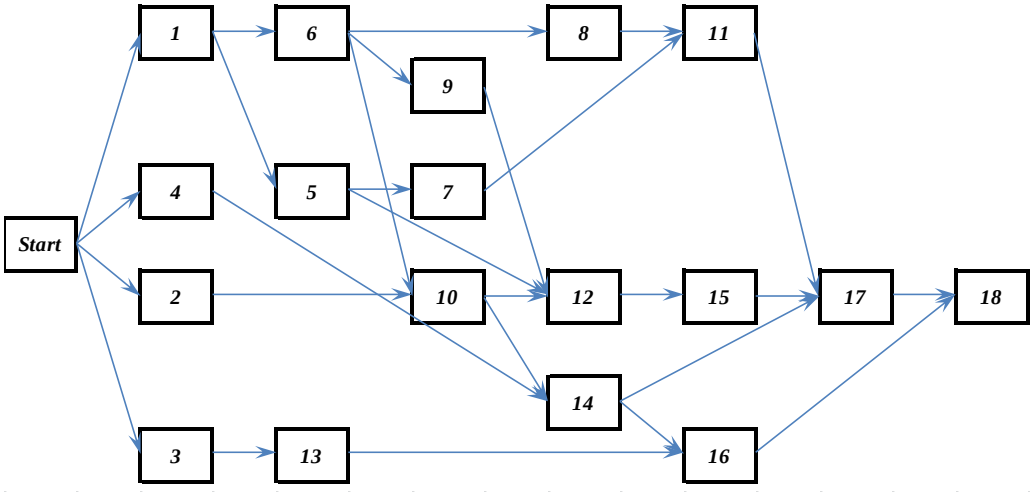


**Figure 3-1 Activity Relationships for the Model Project of 18 Activities**

**Table 3-1 Time and Cost for each Mode of Construction**

| *Activities* | | *3 Methods of Construction - Normal to Crash* | | | | | |
|---|---|---|---|---|---|---|---|
| | | *Cheapest but slowest modes of construction* | | | | *Most expensive but fastest modes of construction* | |
| *Activity ID* | *Name* | *Time 1 (Days)* | *Cost 1 (Dollars)* | *Time 2 (Days)* | *Cost 2 (Dollars)* | *Time 3 (Days)* | *Cost 3 (Dollars)* |
| *1* | *A* | *24* | *1,200* | *21* | *1,500* | *14* | *2,400* |
| *2* | *B* | *25* | *1,000* | *23* | *1,500* | *15* | *3,000* |
| *3* | *C* | *33* | *3,200* | *33* | *3,200* | *15* | *4,500* |
| *4* | *D* | *20* | *30,000* | *20* | *30,000* | *12* | *45,000* |
| *5* | *E* | *30* | *10,000* | *30* | *10,000* | *22* | *20,000* |
| *6* | *F* | *24* | *18,000* | *24* | *18,000* | *14* | *40,000* |
| *7* | *G* | *18* | *22,000* | *18* | *22,000* | *9* | *30,000* |
| *8* | *H* | *24* | *120* | *21* | *208* | *14* | *220* |
| *9* | *I* | *25* | *100* | *23* | *150* | *15* | *300* |
| *10* | *J* | *33* | *320* | *33* | *320* | *15* | *450* |
| *11* | *K* | *20* | *300* | *20* | *300* | *12* | *450* |
| *12* | *L* | *30* | *1,000* | *30* | *1,000* | *22* | *2,000* |
| *13* | *M* | *24* | *1,800* | *24* | *1,800* | *14* | *4,000* |
| *14* | *N* | *18* | *2,200* | *18* | *2,200* | *9* | *3,000* |
| *15* | *O* | *16* | *3,500* | *16* | *3,500* | *12* | *4,500* |
| *16* | *P* | *30* | *1,000* | *28* | *1,500* | *20* | *3,000* |
| *17* | *Q* | *24* | *1,800* | *24* | *1,800* | *14* | *4,000* |
| *18* | *R* | *18* | *2,200* | *18* | *2,200* | *9* | *3,000* |

Duration and cost for the 18-activity project using normal modes of construction (cheapest but slowest modes) are shown in Figure 3-2 and duration and cost using most crashed modes of construction (Most expensive but fastest modes) are presented in Figure 3-3.

| ID | Task Name | Duration | Cost |
|---|---|---|---|
| 0 | **18 Activity Project** | **169 days** | **$99.74** |
| 1 | *1 A* | *24 days* | *$1.20* |
| 2 | *2 B* | *25 days* | *$1.00* |
| 3 | *3 C* | *33 days* | *$3.20* |
| 4 | *4 D* | *20 days* | *$30.00* |
| 5 | *5 E* | *30 days* | *$10.00* |
| 6 | *6 F* | *24 days* | *$18.00* |
| 7 | *7 G* | *18 days* | *$22.00* |
| 8 | *8 H* | *24 days* | *$0.12* |
| 9 | *9 I* | *25 days* | *$0.10* |
| 10 | *10 J* | *33 days* | *$0.32* |
| 11 | *11 K* | *20 days* | *$0.30* |
| 12 | *12 L* | *30 days* | *$1.00* |
| 13 | *13 M* | *24 days* | *$1.80* |
| 14 | *14 N* | *18 days* | *$2.20* |
| 15 | *15 O* | *16 days* | *$3.50* |
| 16 | *16 P* | *30 days* | *$1.00* |
| 17 | *17 Q* | *24 days* | *$1.80* |
| 18 | *18 R* | *18 days* | *$2.20* |

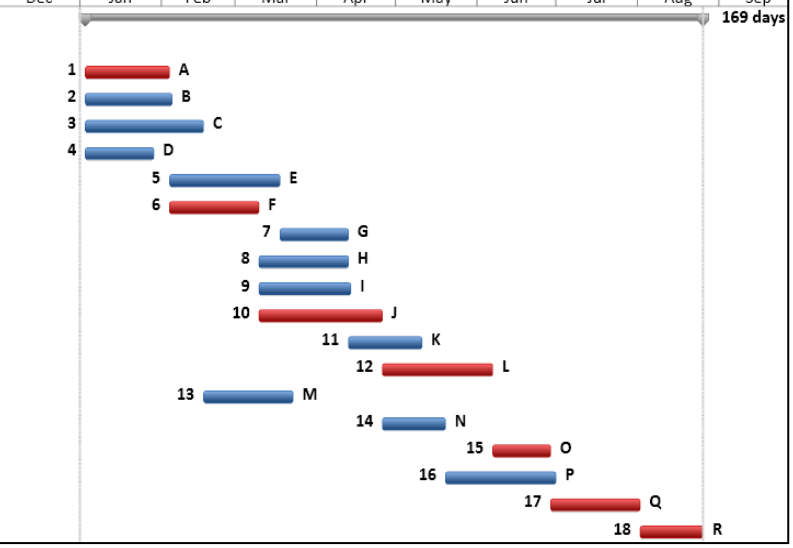


**Figure 3-2 Duration and Cost Using Normal Modes of Construction**

| ID | Task Name | Duration | Cost |
|---|---|---|---|
| 0 | **18 Activity Project** | **100 days** | **$169.82** |
| 1 | *1 A* | *14 days* | *$2.40* |
| 2 | *2 B* | *15 days* | *$3.00* |
| 3 | *3 C* | *15 days* | *$4.50* |
| 4 | *4 D* | *12 days* | *$45.00* |
| 5 | *5 E* | *22 days* | *$20.00* |
| 6 | *6 F* | *14 days* | *$40.00* |
| 7 | *7 G* | *9 days* | *$30.00* |
| 8 | *8 H* | *14 days* | *$0.22* |
| 9 | *9 I* | *15 days* | *$0.30* |
| 10 | *10 J* | *15 days* | *$0.45* |
| 11 | *11 K* | *12 days* | *$0.45* |
| 12 | *12 L* | *22 days* | *$2.00* |
| 13 | *13 M* | *14 days* | *$4.00* |
| 14 | *14 N* | *9 days* | *$3.00* |
| 15 | *15 O* | *12 days* | *$4.50* |
| 16 | *16 P* | *20 days* | *$3.00* |
| 17 | *17 Q* | *14 days* | *$4.00* |
| 18 | *18 R* | *9 days* | *$3.00* |

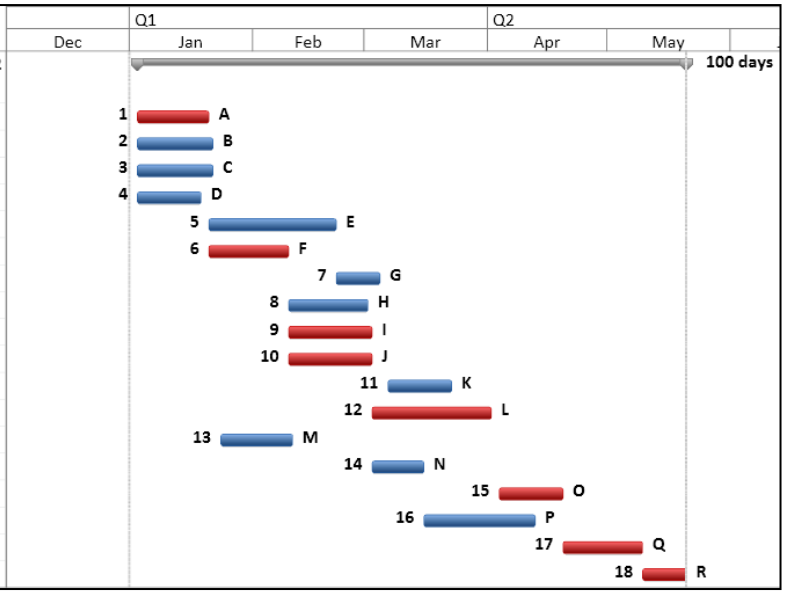


**Figure 3-3 Duration and Cost Using Most Crashed Modes of Construction**

Any optimization method to solve large TCT problems may be more effective with a particular arrangement of activities within a project compared to other arrangements. In other words, the effectiveness of the solution method for a model of a particular size may be

dependent on the relationships of activities in the network. As such, to investigate if this issue affects the optimization results of the research, any experimentation for a particular project size is performed based on both parallel and serial expansion of the 18-activity network.

For example, to optimize a project consisting of 36 activities, two separate experiments are performed: one experiment using the parallel expansion of the 18-activity network and another one using the serial expansion. The sample parallel and serial networks for the project consisting of 36 activities are presented in Figure 3-4 and Figure 3-5 respectively.

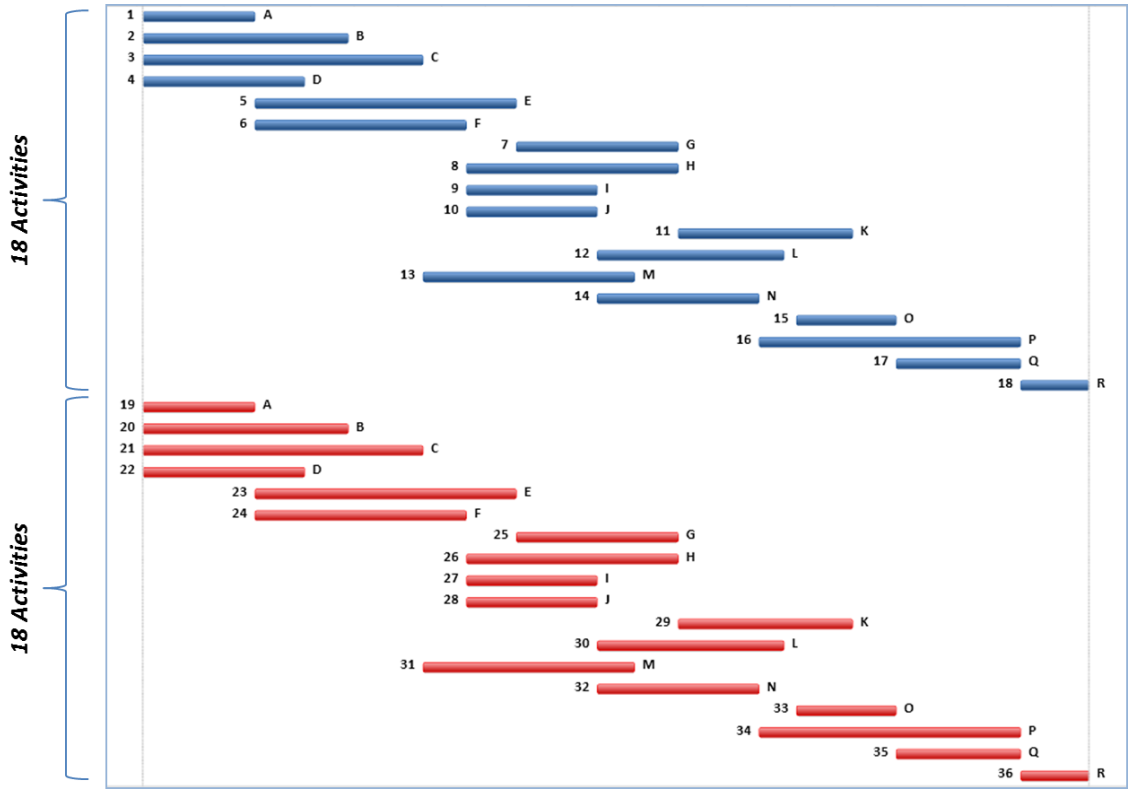


**Figure 3-4 Two Parallel Copies of the 18-Activity Network**

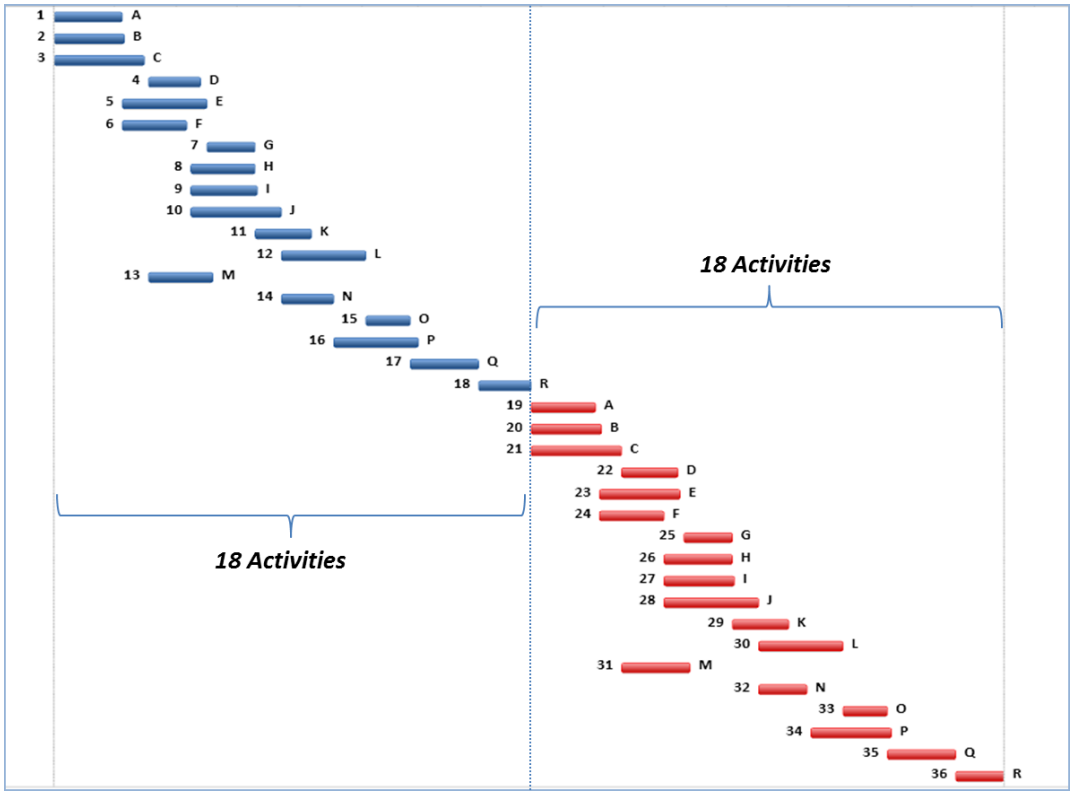


**Figure 3-5 Two Serial Copies of the 18-Activity Network**

## 3.3 Mathematical Optimization: Experimentations and Results

The application of mathematical optimization methods on the TCT problem is investigated using two particular optimization tools. The first optimization tool is Excel Solver which can be used for solving small TCT problems and as a base to compare the results with the results of a more advanced mathematical optimization package. The more advanced modeling and optimization tool is 'Risk Solver Platform' which is capable of solving larger problems. The results are then compared and discussed.

### 3.3.1 TCT Optimization Using Excel Solver

Microsoft Excel includes an optimization utility which is known as Solver Add-in. It is an easy to use optimization tool capable of solving two types of mathematical problems: linear problems and non-linear problems. It solves linear problems using Simplex method, and almost always finds perfect answers to small, linear problems. Microsoft Solver also includes the GRG Nonlinear engine for solving smooth non-linear problems. However, the non-linear routine does not perform as good as the linear algorithm and may stick in a local optimum solution (*The Genetic Algorithm Solver for Microsoft Excel*, 2010). In addition, Solver has limitations of 200 on the number of variables and 200 on the number of constraints and, therefore, is only suitable for solving relatively small problems.

The Structure of the TCT model in Microsoft Excel is based on the model introduced by T. Hegazy & Ayed, 1999. In this model activities are represented in a column and their associated data in the next columns. Therefore, each activity and its corresponding data including its predecessors, successors, and also the duration and cost associated with different possible construction methods (modes) for that particular activity are all in a row. Preferred method of construction for each activity is calculated based on an index which is defined in a column, and associated duration and cost is presented in two other columns on the spreadsheet. A screen-shot of the model is shown in Figure 3-6.

Indirect cost is defined at the bottom of the spreadsheet. Total project duration and cost is then calculated considering total indirect cost. No penalty and incentive have been defined in this model.

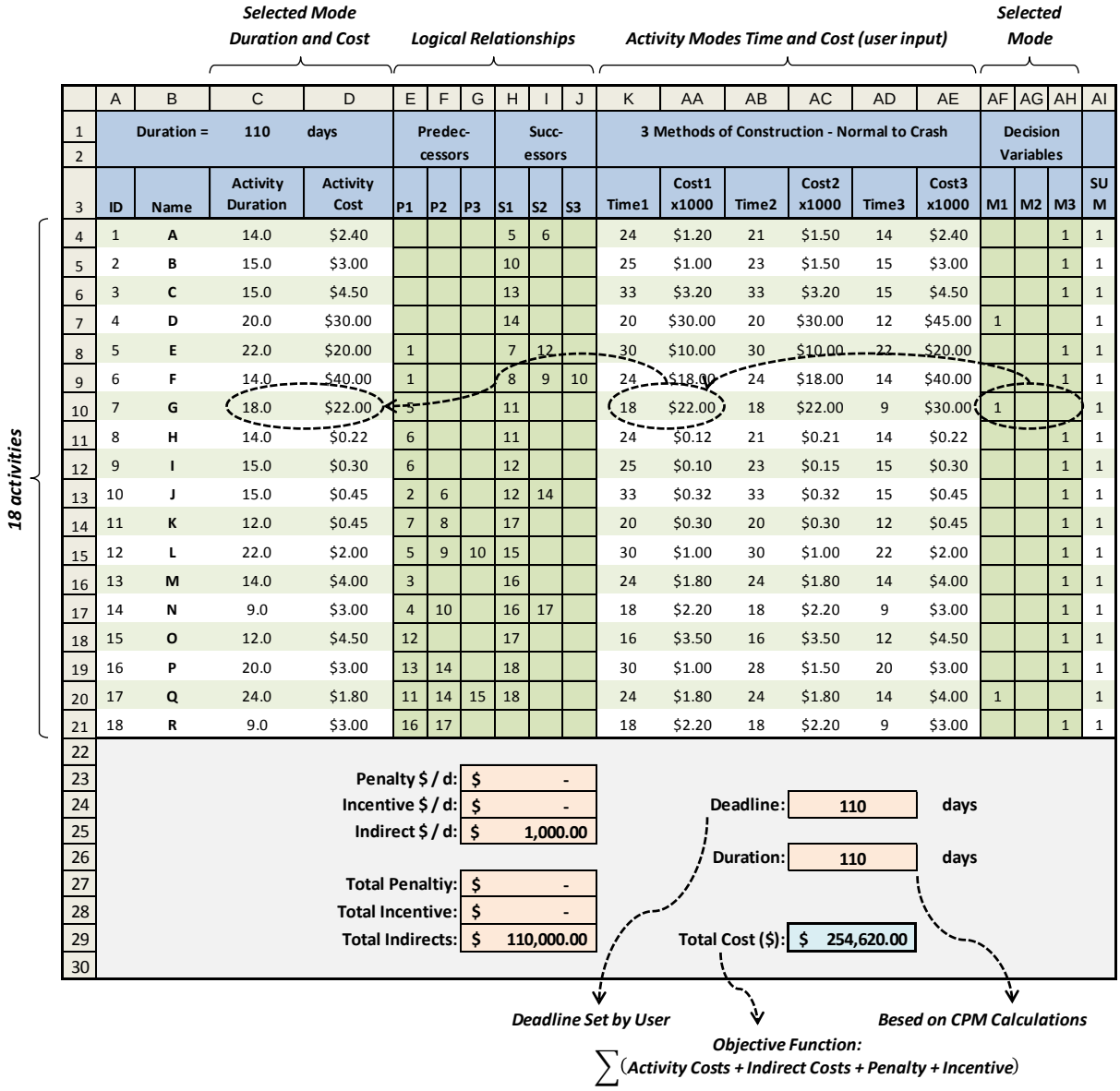


**Figure 3-6 TCT Model Spreadsheet**

The model is a non-linear mathematical model and, therefore, Solver's GRG Nonlinear engine is used to solve it. The objective function is to minimize total cost; decision variables are indexes to choose among different methods of construction, and constraints define the availability of only one method of construction for each activity at any certain time and also

limiting duration to deadline. Maximum amount of calculation time is set to 30 minutes. A snapshot of the solver parameters is shown in Figure 3-7.

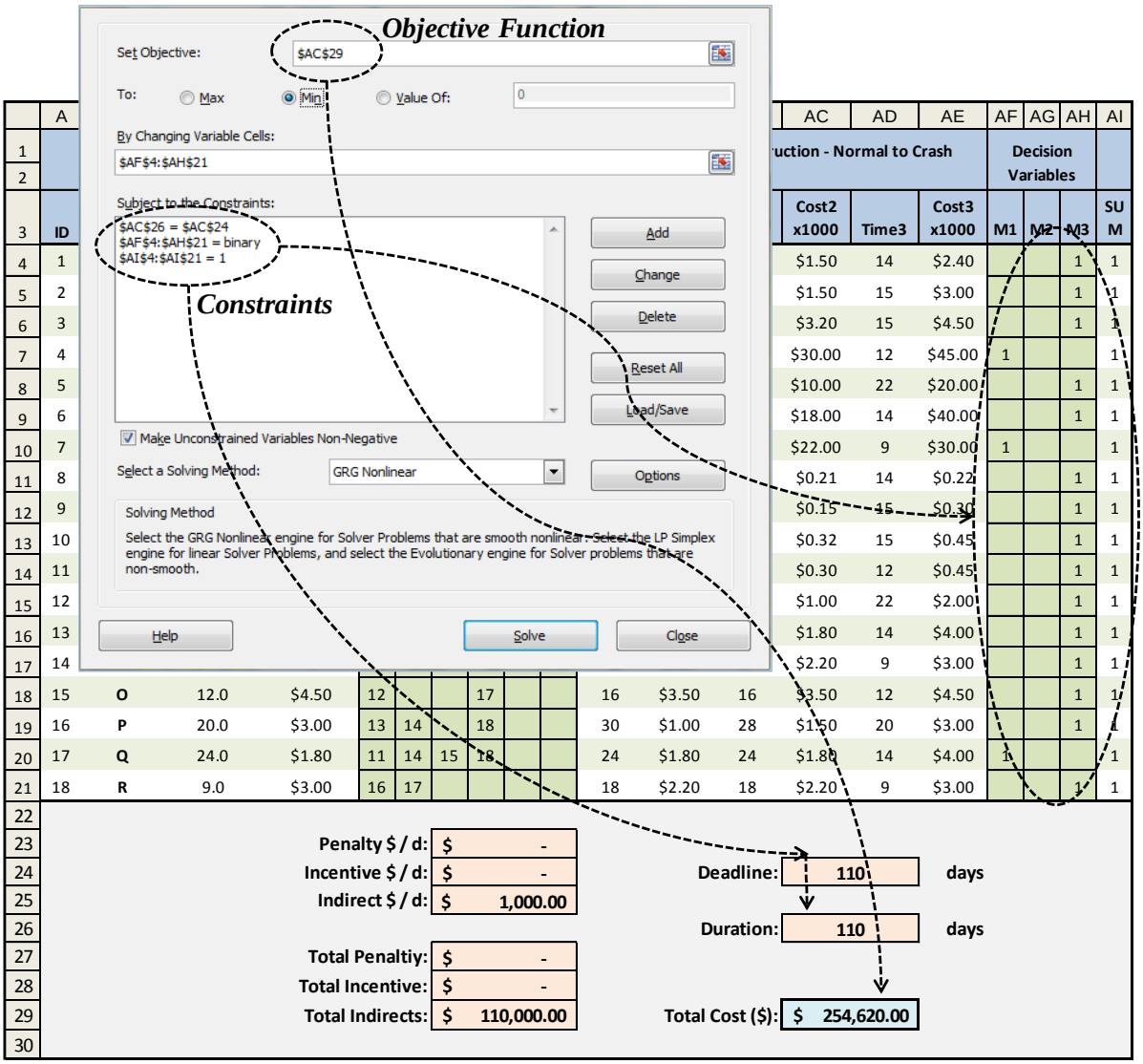


**Figure 3-7 Model Definition in Excel Solver**

The optimization results using Excel Solver for TCT models, ranging from the initial model consisting of 18 activities to models with serial and parallel combinations of the initial model including 36, 54, and 180 activities, are presented in Table 3-2. In addition to the minimum

total cost, Table 3-2 presents the deadline and duration of the project, the optimal solution and the deviation of the result from the optimal solution, and also the calculation time.

**Table 3-2 Optimization Results Using Excel Solver**

| No. of Activities | 18 | 36 P [1] | 36 S [1] | 54 P | 54 S | 180 P | 180 S |
|---|---|---|---|---|---|---|---|
| Project Deadline | 110 | 110 | 220 | 110 | 330 | 110 | 1100 |
| Optimum Solution | 216,270 | 322,540 | 432,540 | 428,810 | 648,810 | 1,172,700 | 2,162,700 |
| Duration (days) | 110 | 110 | 220 | 110 | 330 | | |
| Total Cost (×1000) | 254,620 | 409,840 | 433,094 | 607,435 | 769,460 | N/A | N/A |
| Deviation* | 18% | 27% | 0% | 42% | 19% | Too Many Variable Cells | Too Many Variable Cells |
| Calculation Time (min) | 2 | 5 | 30 | 30 | 13 | | |

1- P: Parallel arrangement of activities; S: Serial arrangement of activities

* Percentage of deviation of the result from optimal solution

A significant advantage of developing larger models based on serial and parallel arrangements of the 18 activities model is that, since the optimum solution of the base model is known, the optimum solution for larger models can simply be calculated independently of the optimization process. Thus, the performance of any optimization can be measured by comparing the deviation of the solutions from the calculated optimum solution.

The number of decision variables for a project consisting of 54 activities each with 3 modes of construction is 54*3=162 and for a project of 180 activities is 540 variables. Therefore,

Excel Solver limitation of 200 decision variables prevents it to optimize TCT models consisting of 180 activities. Excel Solver is only useful for solving small models. In the next section a more advances mathematical optimization tool will be used to solve the same TCT models.

### 3.3.2 TCT Optimization Using Risk Solver Platform (RSP)

Risk Solver Platform is one of the most powerful tools for risk analysis, simulation, and optimization in Microsoft Excel, developed by Frontline Systems Inc. Risk Solver Platform has five bundled Solver Engines and can also be connected to many optional plug-in optimization engines such as Gurobi Solver Engine, MOSEK Solver, and KNITRO Solver (*Frontline Solvers User Guide*, 2010). For the purpose of this research, the nonlinear LSGRG Solver and Large-scale GRG Solver are used to solve TCT problems as a smooth nonlinear (NLP) mathematical problem.

The nonlinear LSGRG Solver in RSP handles problems of up to 1,000 decision variables and 1,000 constraints which is much less restricted than the limitation of 200 decision variables and 200 constraints in Excel Solver. Large-Scale GRG Solver Engine extends the power of Solver Platform to handle smooth nonlinear optimization problems of up to 4,000 variables and 4,000 constraints which enables it to solve larger problems compared to the LSGRG engine.

Since Risk Solver Platform is an Excel-based optimization package, the same TCT model which was discussed in the previous section can be used by Risk Solver Platform. The possibility of using identical model with different optimization tools is a clear advantage to

compare the efficiency and accuracy of different optimization tools in solving identical optimization problems.

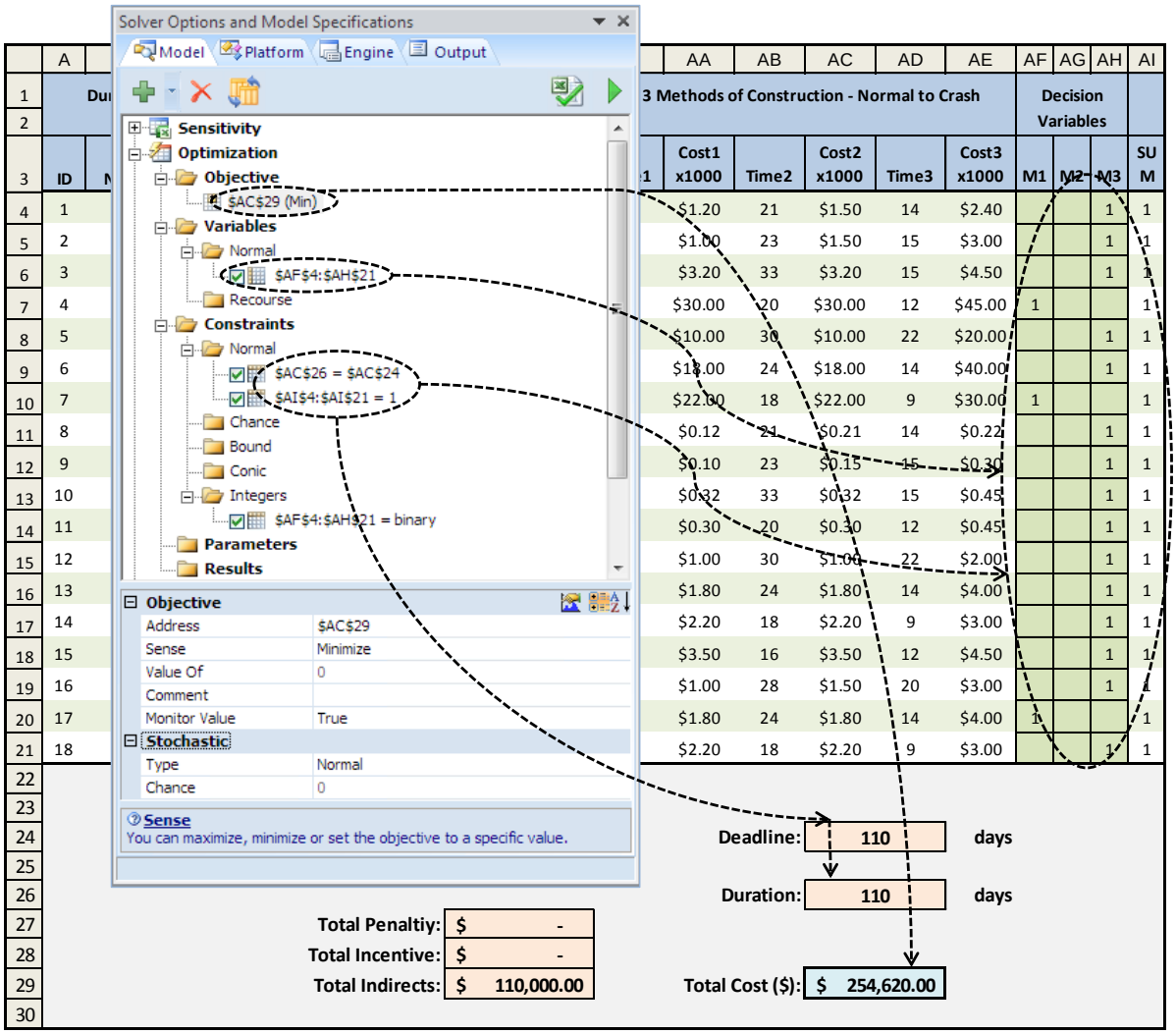


**Figure 3-8 Model Definition in Risk Solver Platform**

The main components of the Excel model in Risk Solver Platform are as follows: 1) The **Objective Function** is defined in the RSP objective section to minimize the total cost. 2) **Decision Variables** are defined in the variable section of RSP. They are in the form of changing cells and are related to data and objective function in the spreadsheet. 3)

**Constraints** are defined in the RSP constraints section. They limit the duration to the deadline and enforce selecting of only one mode of execution for any activity. 4) All of the **data** including activities and their corresponding time and cost for three different modes of construction, as well as, daily direct costs are defined in the main spreadsheet. Maximum computation time is set to 30 minutes. A snapshot of the RSP model definition and its parameters is shown in Figure 3-8.

The optimization results of Risk Solver Platform SLGRG engine for TCT models ranging from the project of 18 activities to models with serial and parallel combinations of the initial model which include 36, 54, and 180 activities, are presented in Table 3-3.

**Table 3-3 Optimization Results Using SLGRG Nonlinear Solver of RSP**

| No. of Activities | 18 | 36 P [1] | 36 S [1] | 54 P | 54 S | 180 P | 180 S |
|---|---|---|---|---|---|---|---|
| Project Deadline | 110 | 110 | 220 | 110 | 330 | 110 | 1100 |
| Optimum Solution | 216,270 | 322,540 | 432,540 | 428,810 | 648,810 | 1,172,700 | 2,162,700 |
| Duration (days) | 110 | 110 | 220 | 110 | 330 | 110 | 1100 |
| Total Cost (×1000) | 216,270 | 385,890 | 478,540 | 454,198 | 698,851 | 1,336,900 | 2,315,071 |
| Deviation* | 0% | 20% | 11% | 6% | 8% | 14% | 7% |
| Calculation Time (min) | 1.5 | 14 | 30 | 26 | 30 | 30 | 30 |

1- P: Parallel arrangement of activities; S: Serial arrangement of activities

* Percentage of deviation of the result from optimal solution

The optimization result for the project consisting of 18 activities is equal to the optimal solution and, therefore, its deviation from the optimal solution is equal to zero. For larger projects, the deviation of the result from the optimal solution is not equal to zero; however, the deviation does not have an increasing trend when the size of the project increases. Risk Solver Platform is not capable of solving projects consisting of 360 activities with 4GB of RAM and initiates an insufficient memory error.

**Table 3-4 Optimization Results Using Large-scale GRG Solver of RSP**

| No. of Activities | 18 | 36 P [(1)] | 36 S [(1)] | 54 P | 54 S | 180 P | 180 S |
|---|---|---|---|---|---|---|---|
| Project Deadline | 110 | 110 | 220 | 110 | 330 | 110 | 1100 |
| Optimum Solution | 216,270 | 322,540 | 432,540 | 428,810 | 648,810 | 1,172,700 | 2,162,700 |
| Duration (days) | 110 | 110 | 220 | 110 | 330 | 110 | 1100 |
| Total Cost (×1000) | 216,270 | 379,990 | 513,369 | 454,198 | 714,551 | 1,583,095 | 2,764,677 |
| Deviation* | 0% | 18% | 19% | 6% | 10% | 35% | 28% |
| Calculation Time (min) | 1.5 | 18 | 30 | 26 | 30 | 30 | 30 |

1- P: Parallel arrangement of activities; S: Serial arrangement of activities

* Percentage of deviation of the result from optimal solution

The optimization results of the same models using Large-scale GRG engine of Risk Solver Platform are presented in Table 3-4. Comparing the results of SLGRG engine and Large-scale GRG engine reveals that Large-scale GRG engine does not perform better than SLGRG

engine especially when the size of the project increases. In addition, similar to the SLGRG engine, the Large-scale GRG engine is not capable of optimizing larger project consisting of 360 activities.

Comparing the results of mathematical optimization using Excel Solver, SLGRG engine, and Large-scale GRG engine of Risk Solver Platform demonstrates that the deviations of the solutions from the optimum solutions for RSP are less than Excel Solver. This is an indication of better performance of RSP comparing to Excel Solver. In addition, SLGRG engine of Risk Solver Platform performs better than the Large-scale GRG engine especially when the size of the project increases. Furthermore, the Excel Solver is not able to solve projects consisting 180 activities and RSP is not able to solve projects consisting of 360 activities and, therefore, none of them are capable of optimizing larger projects such as projects consisting of 360 activities.

### 3.4 Evolutionary-Based Optimization: Experimentations and Results

Due to the complexity of TCT problems and inefficiency of mathematical optimization methods in solving large TCT optimization problems, evolutionary-based optimization methods have been an alternate method for solving time-cost trade-offs.

In this section the performance of two evolutionary-based optimization software packages on solving large TCT problems is examined. First, the evolutionary-based optimization engine in Risk Solver Platform optimization package is examined and then the genetic algorithm modeling and optimization tool, “Evolver”, is used to solve large-scale TCT problems.

### 3.4.1 TCT Optimization Using Risk Solver Platform

Risk Solver Platform includes an Evolutionary Engine implementing evolutionary-based optimization methods for solving problems. Evolutionary Solver is a hybrid engine which contains genetic and evolutionary algorithms and classical optimization methods. It handles non-smooth (NSP) problems of up to 1,000 decision variables and 1,000 constraints. It was greatly enhanced with parallel algorithms and in V11.5 was enhanced with new state-of-the-art methods to help solving hard non-smooth models with better answers in less time (*Frontline Solvers User Guide*, V11.5).

**Table 3-5 Optimization Results Using RSP Evolutionary Engine**

| No. of Activities | 18 | 36 P (1) | 36 S (1) | 54 P | 54 S | 180 P | 180 S |
|---|---|---|---|---|---|---|---|
| Project Deadline | 110 | 110 | 220 | 110 | 330 | 110 | 1100 |
| Optimum Solution | 216,270 | 322,540 | 432,540 | 428,810 | 648,810 | 1,172,700 | 2,162,700 |
| Duration (days) | 110 | 110 | 219 | 110 | 310 | 110 | 1010 |
| Total Cost (×1000) | 275,320 | 438,440 | 566,640 | 618,260 | 1,018,260 | 1,807,000 | 3,607,000 |
| Deviation* | 27% | 36% | 31% | 44% | 57% | 54% | 67% |
| Calculation Time (min) | 18 | 15 | 1 | 1 | 1 | 21 | 30 |

1- P: Parallel arrangement of activities; S: Serial arrangement of activities

* Percentage of deviation of the result from optimal solution

Using Evolutionary Solver of RSP, the model structure and details, remain the same as the model structure and details discussed in Section 3.3.2 using nonlinear LSGRG Solver. The

model definition is also the same as model definition presented in Figure 3-8. The maximum calculation time limit is set to 30 minutes. The optimization results of using evolutionary solver of Risk Solver Platform are displayed in Table 3-5.

The results obtained from this evolutionary engine are not promising. In particular the results for larger problems are far from optimum. In the next section an advanced genetic algorithm optimization tool is used to solve the same sets of TCT problems.

### 3.4.2 TCT Optimization Using Evolver

Evolver is a genetic algorithm optimization add-in for Microsoft Excel developed by Palisade Corporation. Evolver uses innovative genetic algorithm (GA) technology to effectively optimize complex and large-scale optimization problems and has been used to solve many complicated problems in a variety of disciplines. The genetic algorithms used in Evolver use the general concepts of genetic algorithms but have been adapted for Evolver (*The Genetic Algorithm Solver for Microsoft Excel*, 2010). A summary of how evolver works based on the Evolver's Users Guide is represented below:

- **Selection:** When a new organism is to be created, two parents are chosen from the current population. Organisms that have high fitness scores are more likely to be chosen as parents.

- **Crossover:** Since each solving method adjusts the variables in different ways, Evolver employs a different crossover routine optimized for that type of problem.

- **Mutation:** Like crossover, mutation methods are customized for each of the different solving methods. The basic recipe solving method performs mutation by looking at each variable individually. A random number between 0 and 1 is generated for each of the variables in the organism, and if a variable gets a number that is less than or equal to the mutation rate (for example, 0.06), then that variable is mutated.

- **Replacement:** Since Evolver uses a rank-ordered rather than generational replacement method, the worst-performing organisms are always replaced with the new organism that is created by selection, crossover, and mutation, regardless of its fitness "score".

- **Constraints:** Hard constraints are implemented with Palisade's proprietary "backtracking" technology. If a new offspring violates some externally imposed constraints, Evolver backtracks towards one of the parents of the child, changing the child until it falls within the valid solution space.

Since Evolver is an add-on for Microsoft Excel, the same TCT Model which was discussed in Section 3.3.1 is also used for experimentations using Evolver. The main components of the model including objective function, decision variables and, constraints are defined in the model definition window of Evolver.

Table 3-6 depicts the optimization results of using Evolver for TCT models ranging from the network of 18 activities to models consisting of 36, 54, and 180 activities.

**Table 3-6 Optimization Results Using Evolver**

| No. of Activities | 18 | 36 P (1) | 36 S (1) | 54 P | 54 S | 180 P | 180 S |
|---|---|---|---|---|---|---|---|
| Project Deadline | 110 | 110 | 220 | 110 | 330 | 110 | 1100 |
| Optimum Solution | 216,270 | 322,540 | 432,540 | 428,810 | 648,810 | 1,172,700 | 2,162,700 |
| Duration (days) | 110 | 109 | 220 | 110 | 330 | 109 | 1063 |
| Total Cost (×1000) | 238,070 | 373,790 | 484,640 | 500,610 | 700,410 | 1,801,700 | 3,087,390 |
| Deviation* | 10% | 16% | 12% | 17% | 8% | 54% | 43% |
| Calculation Time (min) | 30 | 30 | 30 | 30 | 30 | 30 | 30 |

1- P: Parallel arrangement of activities; S: Serial arrangement of activities

* Percentage of deviation of the result from optimal solution

In solving TCT problems using Evolver, the initial values of variables are extremely important. When the optimization starts with a combination of activity modes that conclude to project duration of more than defined deadline (infeasible solution), the optimization procedure spends a lot of time to find a feasible solution and then starts improving it. To eliminate this initial processing time the models have been defined such that the optimization starts with the most crashed activity modes as the initial values. This definition proved to be very helpful in reducing optimization time and is used for all of the experiments with evolver and Risk Solver Platform.

As Table 3-6 reveals, the optimization results of using Evolver are also far from optimal. Since evolutionary-based optimization methods may require considerable amount of time to converge to a near optimal solution, the experiments for project networks of 180 activities has been repeated for extended periods of calculation time to investigate whether they reach to a better solution.

**Table 3-7 Evolver Results for Extended Calculation Times of 12h and 24h**

| No. of Activities | 180 P (1) | 180 S (1) | No. of Activities | 180 Act P | 180 Act S |
|---|---|---|---|---|---|
| Project Deadline | 110 | 1100 | Project Deadline | 110 | 1100 |
| Optimum Solution | 1,172,700 | 2,162,700 | Optimum Solution | 1,172,700 | 2,162,700 |
| Duration (days) | 110 | 1100 | Duration (days) | 110 | 1100 |
| Total Cost (×1000) | 1,396,688 | 2,441,088 | Total Cost (*1000) | 1,389,500 | 2,436,300 |
| Deviation* | 19% | 13% | Deviation* | 18% | 13% |
| Calculation Time (min) | 720 | 720 | Calculation Time (min) | 1440 | 1440 |

1- P: Parallel arrangement of activities; S: Serial arrangement of activities

* Percentage of deviation of the result from optimal solution

The experiments for 12 hours and 24 hours processing times are shown in Table 3-7. These experiments represent 30%-35% improvements in 12 hours processing times comparing to 30 minutes, and almost no improvement for 24 hours processing time comparing to 12 hours calculation times.

## 3.5 Summary and Conclusions

In this chapter mathematical and evolutionary-based optimization methods which are called the traditional optimization methods for solving time-cost trade-off problems were examined for solving TCT problems. Mathematical methods were implemented by Excel Solver and Risk Solver Platform. Evolutionary-based algorithms were implemented using Risk Solver Platform and Evolver. The summary of optimization results are shown is Table 3-8.

Finding an initial feasible solution for a TCT problem is an important step for solving it while using mathematical or evolutionary-based optimization methods. In some cases it takes a considerable amount of time for the optimization process to find a feasible solution, and then start to improve it. To eliminate this lengthy part of the processing time, the values of the decision variables for mathematical and evolutionary-based optimization methods were initialized to start the optimization process with a feasible solution.

Table 3-8 clearly shows that utilizing traditional optimization methods for solving large TCT problems, leads to solutions far from the optimal solution. Experiments using genetic algorithm method with larger processing times such as 12 and 24 hours also revealed that, large processing times improve the solution to some extent but after extended periods of processing time, the optimization process sticks to a local optimum which is still far from the optimal solution.

**Table 3-8 Comparison of Mathematical and Evolutionary-Based Optimization Results**

| | No. of Activities | 18 | 36 P [1] | 36 S [1] | 54 P | 54 S | 180 P | 180 S |
|---|---|---|---|---|---|---|---|---|
| | Project Deadline | 110 | 110 | 220 | 110 | 330 | 110 | 1100 |
| | Optimum Solution | 216,270 | 322,540 | 432,540 | 428,810 | 648,810 | 1,172,700 | 2,162,700 |
| Excel Solver | Duration (days) | 110 | 110 | 220 | 110 | 330 | | |
| | Total Cost (×1000) | 254,620 | 409,840 | 433,094 | 607,435 | 769,460 | N/A | N/A |
| | Deviation* | 18% | 27% | 0% | 42% | 19% | Too Many Variable Cells | Too Many Variable Cells |
| | Calculation Time (min) | 2 | 5 | 30 | 30 | 13 | | |
| Risk Solver Platform Standard SLGRG Nonlinear | Duration (days) | 110 | 110 | 220 | 110 | 330 | 110 | 1100 |
| | Total Cost (×1000) | 216,270 | 385,890 | 478,540 | 454,198 | 698,851 | 1,336,900 | 2,315,071 |
| | Deviation* | 0% | 20% | 11% | 6% | 8% | 14% | 7% |
| | Calculation Time (min) | 1.5 | 14 | 30 | 26 | 30 | 30 | 30 |
| Risk Solver Platform Large-scale GRG Solver | Duration (days) | 110 | 110 | 220 | 110 | 330 | 110 | 1100 |
| | Total Cost (×1000) | 216,270 | 379,990 | 513,369 | 454,198 | 714,551 | 1,583,095 | 2,764,677 |
| | Deviation* | 0% | 18% | 19% | 6% | 10% | 35% | 28% |
| | Calculation Time (min) | 1.5 | 18 | 30 | 26 | 30 | 30 | 30 |

1- P: Parallel arrangement of activities; S: Serial arrangement of activities

* Percentage of deviation of the result from optimal solution

**Table 3-8 Continued.**

| | No. of Activities | 18 | 36 P [1] | 36 S [1] | 54 P | 54 S | 180 P | 180 S |
|---|---|---|---|---|---|---|---|---|
| | Project Deadline | 110 | 110 | 220 | 110 | 330 | 110 | 1100 |
| | Optimum Solution | 216,270 | 322,540 | 432,540 | 428,810 | 648,810 | 1,172,700 | 2,162,700 |
| Risk Solver Platform Standard Evolutionary Solver | Duration (days) | 110 | 110 | 219 | 110 | 310 | 110 | 1010 |
| | Total Cost (×1000) | 275,320 | 438,440 | 566,640 | 618,260 | 1,018,260 | 1,807,000 | 3,607,000 |
| | Deviation* | 27% | 36% | 31% | 44% | 57% | 54% | 67% |
| | Calculation Time (min) | 18 | 15 | 1 | 1 | 1 | 21 | 30 |
| Evolver | Duration (days) | 110 | 109 | 220 | 110 | 330 | 109 | 1063 |
| | Total Cost (×1000) | 238,070 | 373,790 | 484,640 | 500,610 | 700,410 | 1,801,700 | 3,087,390 |
| | Deviation* | 10% | 16% | 12% | 17% | 8% | 54% | 43% |
| | Calculation Time (min) | 30 | 30 | 30 | 30 | 30 | 30 | 30 |

1- P: Parallel arrangement of activities; S: Serial arrangement of activities

* Percentage of deviation of the result from optimal solution

In the next chapter a new promising optimization method is investigated. This new method may have the potential to provide better results in less time, worth to be applied to larger time-cost trade-off problems.

# Chapter 4
# TCT Optimization Using Constraint Programming

## 4.1 An Introduction to Constraint Programming

Constraint programming is a powerful technology for solving large-scale combinatorial problems. It is based on a broad range of techniques from artificial intelligence, operations research, applied mathematics, and programming languages, to computer science fundamentals such as logic programming and graph theory (Dechter, 2003). Recent advancements in the development of tunable and robust constraint programming search engines has turned this technology into a powerful and efficient optimization technology (*IBM ILOG Optimization Studio V12.2 - Language User's Manual*). Constraint programming is currently applied successfully to many fields, such as scheduling, planning, vehicle routing, networks, and bioinformatics (Rossi, Beek, & Walsh, 2006).

The Constraint Satisfaction Problem (CSP) is the main concept in the heart of constraint programming. A constraint satisfaction problem consists of a set of variables with well-defined domains of possible values and a set of constraints. In addition, some CSPs have an objective function to be minimized or maximized. Any feasible solution to the CSP is an assignment of acceptable values to all of the variables in such a way that satisfies all the constraints. CSPs with an objective function turn the feasibility problem to an optimization problem (Pinedo, 2009), in which the optimal solution, is the feasible solution minimizing (or maximizing) the objective function.

The principle technique for solving constraint satisfaction problems is search. A search method for solving CSP problems may be a complete or an incomplete search. A complete or systematic search method guarantees finding a solution, if one exists, and can also be used to show that there is no solution to the problem. In addition, a complete search method provides the proof of optimality for the solution. Incomplete or non-systematic search methods, on the other hand, cannot provide the proof of optimality and cannot be used to show that there is no solution to the problem. Incomplete search methods, however, are usually efficient in finding a near optimal solution if one exists (Rossi et al., 2006).

Many real world combinatorial problems can be modeled and solved with constraint programming techniques. Constraint programming is a promising method for solving TCT problems. In this chapter we use IBM ILOG Optimization Studio to implement constraint programming techniques for solving TCT problems.

## 4.2 Mathematical Programming vs. Constraint Programming

Regardless of the size, if a problem can be modeled as a Linear Problem (LP), the best method for solving it would be the mathematical LP techniques. However, when it comes to nonlinear, non-smooth problems particularly with discrete variables, the mathematical methods are not so efficient. Furthermore, when it comes to combinatorial problems in which the size grows exponentially, constraint programming models may be much more efficient than mathematical models. The main modeling differences and optimization engine differences for mathematical programming and constraint programming techniques are as follows (*IBM ILOG Optimization Studio V12.4 - Language User's Manual*):

- Mathematical programming models support both continuous and discrete decision variables whereas constraint programming models support only discrete variables (Boolean or Integer).

- A mathematical model should fall in a particular category of problems with certain type of formulation such as linear, convex, integer, quadratic, or nonlinear problem to be solved by mathematical optimization methods; however, a constraint programming model has no limitation on the relation of variables within the constraints and objective function.

- A constraint programming model supports logical constraints and some specialized constraints, such as the "all-different" constraint, that can improve the search speed for finding the solution.

- The solution methods of mathematical programming and constraint programming are fundamentally different. A mathematical programming engine uses mathematical techniques such as simplex method, branch and bound, cutting planes in addition to applying some relaxations to solve the problem. The constraint programming engine; however, assigns values to some variables and based on some logical inferences reduces the domains of the other variables, and then based on some search algorithms finds the solution.

- A mathematical programming engine reduces the feasible region and uses the gap between the lower and upper bound for the feasible region to prove the optimality of

the solution. A constraint programming engine, however, provides the optimality proof by showing that no better solution than the current one can be found.

Table 4-1 presents a summary of differences between mathematical programming and constraint programming.

**Table 4-1 Comparison of Mathematical Programming vs. Constraint Programming (*IBM ILOG Optimization Studio V12.2 - Language User's Manual*)**

| Feature | Mathematical Ptogramming | Constraint Programming |
|---|---|---|
| **Relaxation** | *Yes* | *No* |
| **Optimality GAP measure** | *Yes* | *No* |
| **Optimality proof** | *Yes* | *Yes* |
| **Modeling limitations** | *Model should be in a predefined mathematical category* | *Discrete problems* |
| **Specialized constraints** | *No* | *Yes* |
| **Logical constraints** | *Yes* | *Yes* |
| **Theoretical grounds** | *Algebra* | *Graph theory and algorithmic* |
| **Modeler support** | *Yes* | *Yes* |
| **Model and run** | *Yes* | *Yes* |

### 4.3 IBM ILOG CPLEX Optimization Studio

IBM ILOG CPLEX Optimization Studio is one of the leading optimization software packages for mathematical programming as well as constraint programming. The term CPLEX is an acronym of Simplex method, the well-known linear programming technique, in C programming language. CPLEX was first offered commercially in 1988 by CPLEX Optimization Inc. which was acquired later by ILOG Company in 1997. In 2009, ILOG was acquired by IBM and CPLEX is now part of IBM ILOG Optimization Studio.

IBM ILOG Optimization Studio is a consolidation of some previously stand-alone packages in a single product. It consists of 1) IBM ILOG CPLEX Optimizer for mathematical optimization, 2) IBM ILOG CP Optimizer for constraint programming optimization, 3) the optimization programming language (OPL) for modeling, and 4) an Integrated Development Environment (IDE). CPLEX Optimization Studio provides an excellent way to build efficient optimization models and applications for mathematical and constraint programming problems.

CPLEX Optimizer supports the mathematical algorithms for solving linear problems, mixed-integer problems, quadratic and quadratically constrained problems. CP Optimizer supports all the required constructs for scheduling and time-tabling problems, including tasks, activities and resources. The CPLEX Optimizer has a number of modeling interfaces to C++, C#, .net, Java, and Python languages and includes connectors to Microsoft Excel and MATLAB. It is also accessible with full compatibility through most of the other popular

optimization modeling tools, such as AIMMS, AMPL, GAMS, MPL and Microsoft Solver Foundation.

### 4.3.1 Integrated Development Environment (IDE)

The integrated development environment is the place to create a project, develop and edit the model using OPL language and IBM ILOG Script for OPL, utilize external data, browse and explore different parts of the model, run and debug and tune the model, and finally generate the results and inspect the solution. A snapshot of the IBM ILOG Optimization Studio IDE is shown in Figure 4-1.

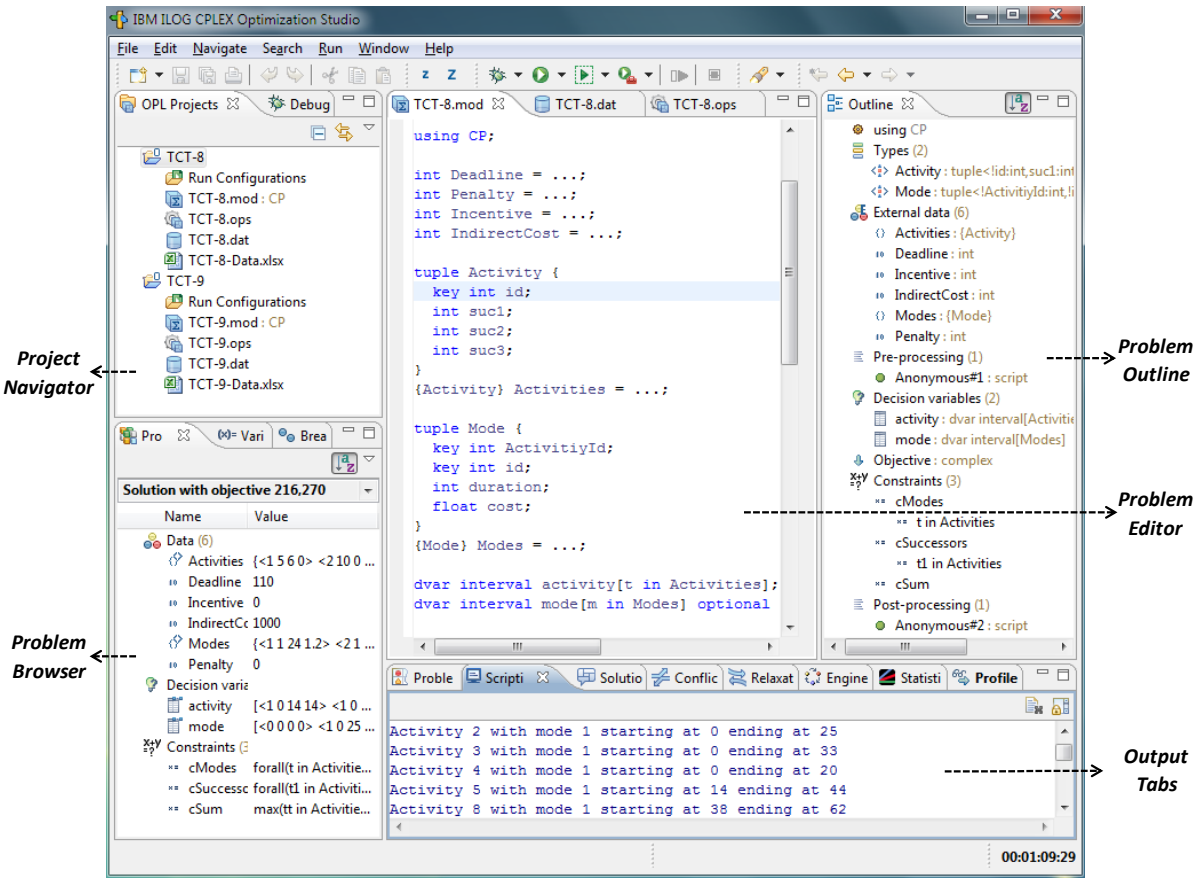


**Figure 4-1 IBM ILOG Optimization Studio IDE**

### 4.3.2 Optimization Programming Language (OPL)

The Optimization Programming Language is a modeling language for mathematical and constraint programming optimization problems which in particular simplifies the formulation and solution of combinatorial optimization problems. As a modeling language, OPL provides high-level specifications, set notations and data modeling facilities to decrease the modeling time of optimization problems (P. Hentenryck & Michel, 2000). The most significant aspect of the OPL, however, is the support for constraint programming, which includes sophisticated search specifications, logical and higher order constraints, and support for scheduling and resource allocation applications (P. V. Hentenryck, 1999).

Mathematical modeling languages such as AMPL and GAMS provide high-level algebraic and set notations to model mathematical problems very efficiently which can then be solved using state-of-the-art solvers (P. Hentenryck, Michel, Laborie, Nuijten, & Rogerie, 1999). The optimization programming language (OPL), however, is the first modeling language that combined the strengths of mathematical modeling languages with the rich constraint language and the ability to specify search procedures and strategies that is the essence of constraint programming (P. Hentenryck, Michel, Perron, & Régin, 1999) both at the language and at the solver levels.

Within the IBM® ILOG® OPL product, OPL has been redesigned to better accommodates its associated Script language. IBM ILOG Script for OPL is a script language for non-modeling aspects such as flow control, pre-processing, and post-processing (*IBM ILOG OPL Language Reference Manual*).

### 4.3.3 OPL Projects in IBM ILOG Optimization Studio

A project in the IBM ILOG Optimization Studio has the following file components:

1. **Model files:** Model (.mod) files contain all the OPL statements
2. **Data files:** Data (.dat) files are used for separating the model from the instance data
3. **Settings files:** Settings files (.ops) are used to change the default values of options and parameters
4. **Run configurations:** Are used to support multiple execution configurations.

A project is the method used to associate a model (.mod) file with, usually, one or more data (.dat) files and one or more settings (.ops) files. A project may include only a single model file or may contain several sets of model, data and settings files.

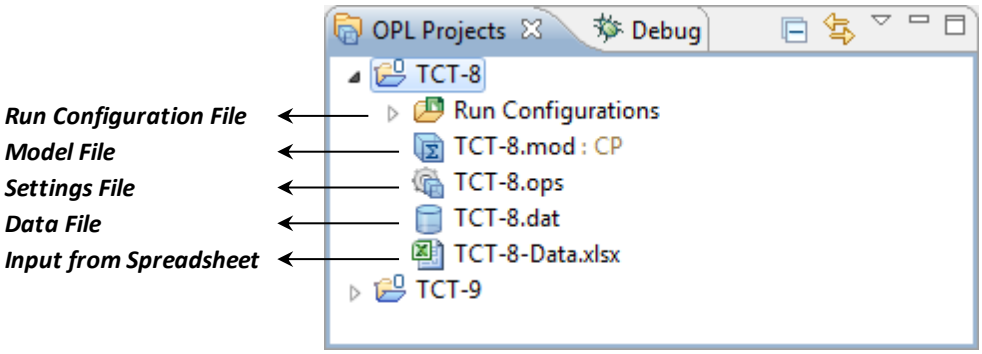


**Figure 4-2 OPL Project Navigator Tab for TCT Model**

Run configurations files define the relationships between model, data and settings files. The model file declares data elements but does not necessarily initialize them. The data files contain the initialization of data elements declared in the model. The .project file in the root

folder for the OPL project organizes all the related model, data and settings files. Run configurations, which are maintained in an .opl project file, also provide a convenient way to maintain the relationship between related files and runtime options for the environment (*IBM ILOG Optimization Studio V12.2 - Language User's Manual*).

The OPL Project Navigator is the tab for managing projects and for creating, adding and removing their associated files such as model, data and settings files. A snapshot of the Project Navigator Tab is shown in Figure 4-2.

## 4.4 Modeling TCT Problems Using IBM ILOG Optimization Studio

The CPLEX Optimizer, the mathematical engine of IBM ILOG Optimization Studio is accessible through independent modeling systems, such as AIMMS, AMPL, GAMS, MPL, and also through a connector to Microsoft Excel. However, the CPLEX CP Optimizer which is the constraint programming engine of IBM ILOG Optimization Studio utilizes specialized data structures and syntax for formulating detailed scheduling models which are only available in OPL language. Therefore, for using constraint programming techniques, the modeling should be implemented in OPL language.

The TCT model of this research in IBM ILOG Optimization Studio is defined as a project consisting of the following components:

- One model (.mod) file containing all the OPL statements for the model definition;
- One data (.dat) file providing the connection to a spreadsheet to access the external data;

- One settings (.ops) file to specify the CP engine parameters and settings;
- One run configuration file referencing and maintaining the relationship between the model, the data and the settings files;
- And a spreadsheet where all the external data are stored.

Figure 4-2 is a snapshot of the OPL Project Navigator tab for the TCT model presenting the file components of the projects in this research.

The model file contains declaration of data, declaration of decision variables, the objective function, and constraints written in OPL language. In addition, it includes some pre- and post-processing statements in OPL scripting language. Details of the TCT model definition sections in OPL language are as follows:

- **Declarations of Data** in the model file are references to data (.dat) file in order to separate data from the model. In large problems separation of the data from the model is a significant advantage for better organization of the model. These references include the number of activities in the project, predecessors and successors of each activity, deadline of the project, penalty and incentive values for late and early completion, indirect cost per day, as well as duration and cost for different modes of each activity. Deadline and Indirect-Cost are defined as integer values, while Activities and Modes are defined as sets of tuples, a combination of different data structures.

- **Decision Variables** are the modes of construction for each activity so that one mode per activity will be selected in the optimization process in order to satisfy the objective function and constraints. In the model decision variables are defined by the "dvar" keyword of the OPL language.

- **Objective function** is to minimize the total cost for the project based on the selection of modes of construction for each activity. It is the sum of direct cost, penalty and incentive for the duration of the project. The objective function in the model is defined by the OPL keyword "Minimize".

- **Constraints** in the model file define all the relationships (predecessors and successors) between activities and limit the total project duration to the deadline. Constraints are defined with "subject to {}" keyword comprising a block of OPL statements.

The pre-processing and post-processing OPL scripting language statements are defined with the keyword "execute {}". The pre-processing block limits the maximum allowable processing time to 30 minutes and the maximum allowable number of failed branches to 1E+6 branches. The post-processing block presents the details of the final solution including the information regarding the selected mode for each activity and the total project cost. Figure 4-3 displays part of the model file which includes the data declarations for modes of construction, declaration of decision variables and also the pre-processing script statements. A complete source code of the model file is included in Appendix B.

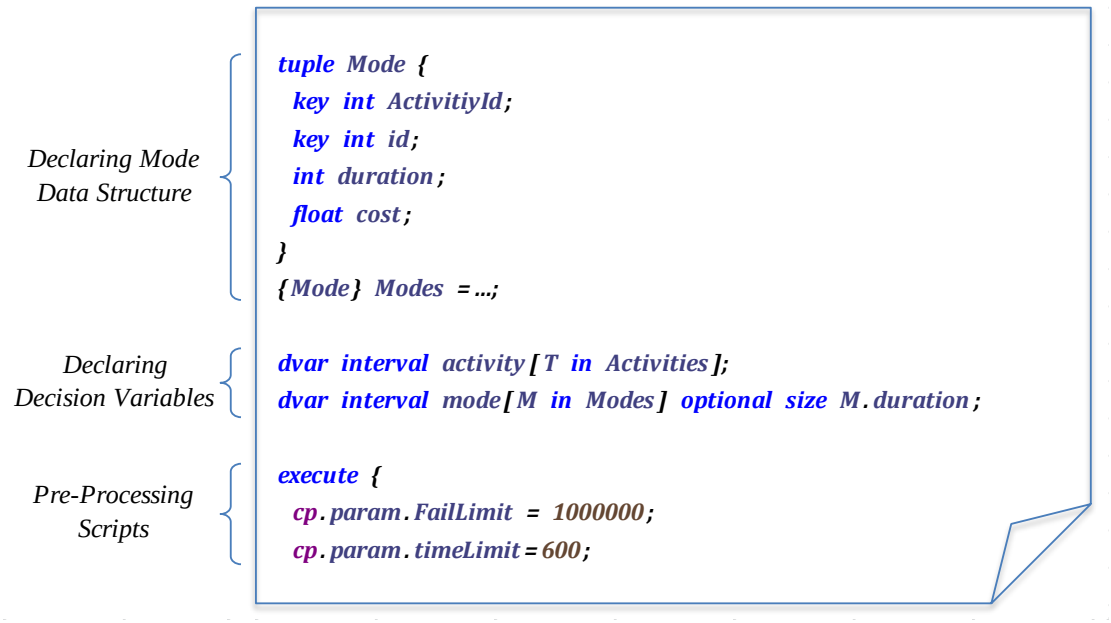


**Figure 4-3 Part of Model (Mod.) File of TCT Model**

The data file is where all the external data are defined. One possibility is to define all the external data explicitly in the data file. The other option is to define a connection within the data file to a database or a spreadsheet where actual data can be retrieved. Since data preparation for various projects and their corresponding activities and modes of construction within each activity is much more convenient in a spreadsheet or a database comparing to data preparation within a data file, the second option is more appropriate for this research. Consequently, the data file provides the references to spreadsheets in which the actual external data can be accessed. Each spreadsheet contains all the information for one project to optimize one instance of the TCT problem. The actual data includes the number of activities in the project, predecessors and successors of each activity, project deadline, indirect cost per day, and also duration and cost for different modes of each activity.

Figure 4-4 displays the data file and the references to a spreadsheet where the actual data will be retrieved.

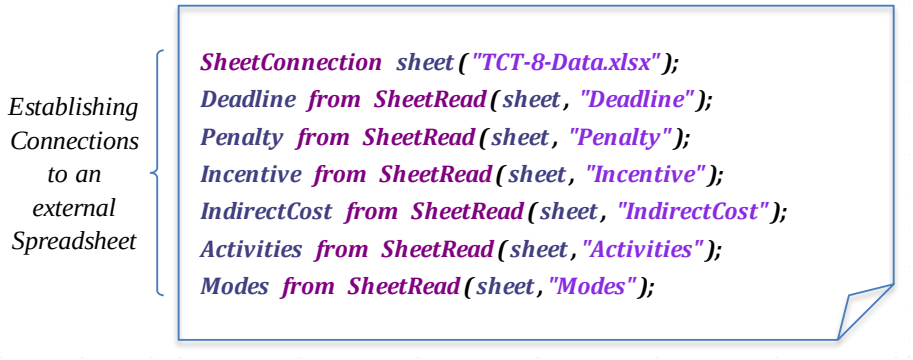


**Figure 4-4 Data (.dat) File of TCT Model**

Figure 4-5 is a snapshot of the contents of settings file (.ops) where the default values of options and parameters can be changed. These changes are stored in the settings (.ops) file and control various parameters for the solution process.

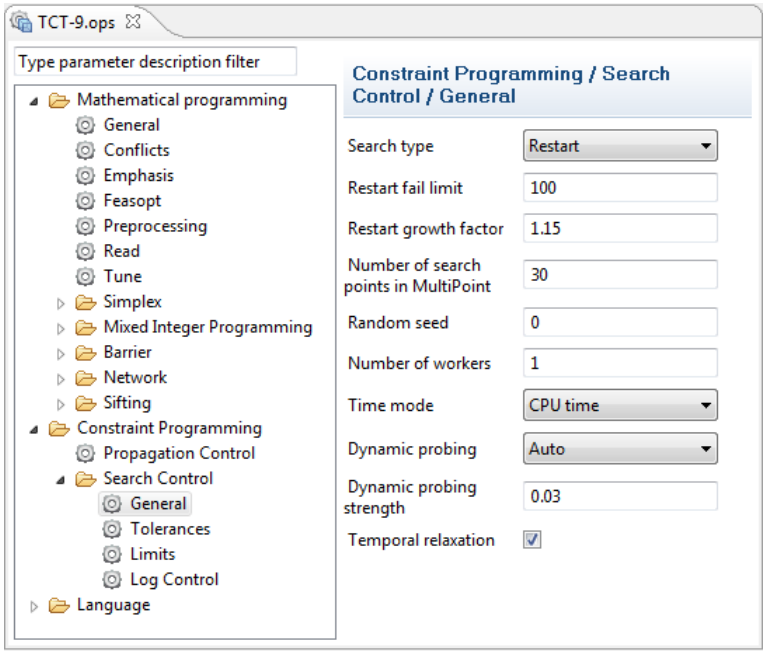


**Figure 4-5 Settings (.ops) File of TCT Model**

## 4.5 Constraint Programming Optimization: Experimentations and Results

Once a TCT project in the CPLEX Studio environment is executed, the solution steps are presented in the "Engine log" and the "Statistics" tabs. At the beginning of the optimization, the engine log displays the initial process time and the memory usage. Throughout the optimization process it shows the number of branches, the branch decisions, and the current best solution. At the end of the optimization, it presents the total processing time, the final solution, and the total memory usage. Figure 4-6 shows part of the report of the engine log at the beginning of the optimization.

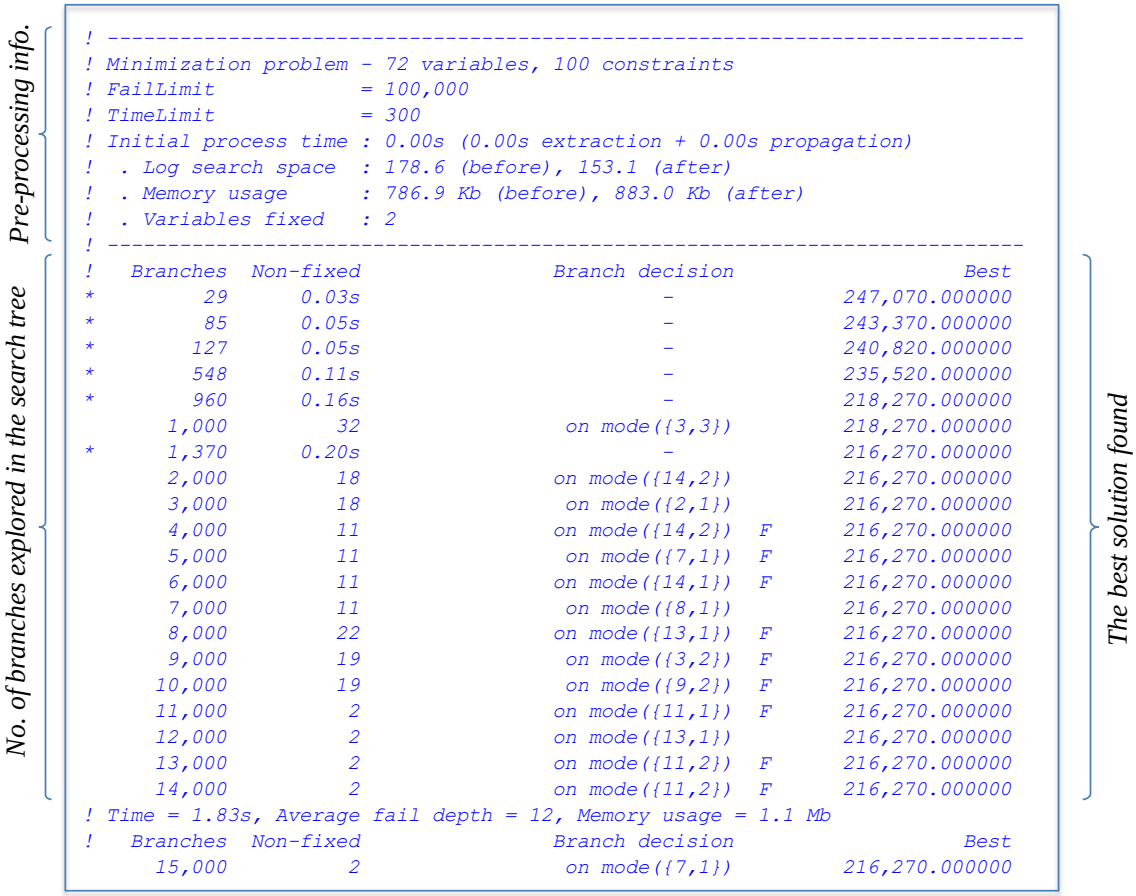


**Figure 4-6 Part of the Engine Log Report for the 18-Activity Project**

In addition, during the optimization, the statistics tab visualizes the optimization process by drawing a graph representing the improvements of the solution according to a time-scale in seconds. Figure 4-7 shoes a snapshot of statistics tab for solving a project consisting of 3600 activities using CP Optimizer.

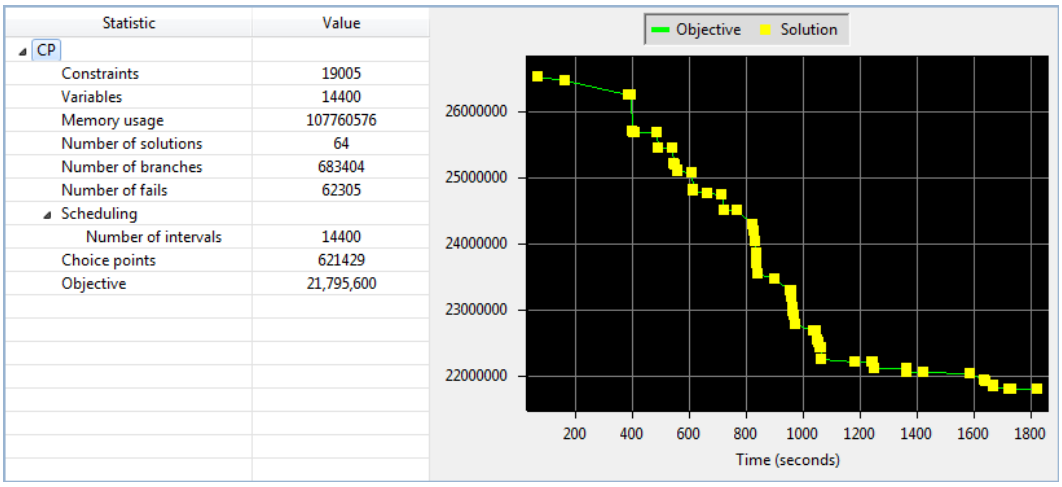


**Figure 4-7 Sample Statistics for Solving a Project of 3600 Activities Using CP Optimizer**

The experimentations for solving TCT problems using CP Optimizer are performed for the base model of 18 activities, as well as, serial and parallel arrangements of the base model consisting of 36, 54, and 180, 1800, 3600, 5400, and 10800 activities.

The optimization results are presented in Table 4-2. In addition to the final solution (total cost) for each experiment, Table 4-2 presents the calculation time, the duration of the project after optimization, and the deviation of the final solution from the optimum solution. In these experiments no penalty and incentive has defined to be compared with the optimization results of previous chapters.

**Table 4-2 Optimization Results using CPLEX CP Optimizer**

| No. of Activities | 18 | 36 P [1] | 36 S [1] | 54 P | 54 S | 180 P | 180 S |
|---|---|---|---|---|---|---|---|
| Project Deadline | 110 | 110 | 220 | 110 | 330 | 110 | 1100 |
| Optimum Solution | 216,270 | 322,540 | 432,540 | 428,810 | 648,810 | 1,172,700 | 2,162,700 |
| Duration (days) | 110 | 110 | 220 | 110 | 330 | 110 | 1100 |
| Total Cost (×1000) | 216,270 | 322,540 | 432,540 | 428,810 | 648,810 | 1,172,700 | 2,183,700 |
| Deviation* | 0% | 0% | 0% | 0% | 0% | 0% | 1% |
| Calculation Time (min) | 9 | 10 | 10 | 10 | 10 | 13 | 14 |

| No. of Activities | 1800 P | 1800 S | 3600 P | 3600 S | 5400 P | 5400 S | 10800 P | 10800 S |
|---|---|---|---|---|---|---|---|---|
| Project Deadline | 110 | 11000 | 110 | 22000 | 110 | 33000 | 110 | 66000 |
| Optimum Solution | 10,737,000 | 21,627,000 | 21,364,000 | 43,254,000 | 31,991,000 | 64,881,000 | 63,872,000 | 129,762,000 |
| Duration (days) | 110 | 11000 | 110 | 22000 | 110 | 33000 | 110 | 66000 |
| Total Cost (×1000) | 10,737,000 | 23,592,960 | 22,031,750 | 47,121,800 | 33,593,600 | 73,272,560 | 82,352,000 | 146,615,840 |
| Deviation* | 0% | 9% | 3% | 9% | 5% | 13% | 29% | 13% |
| Calculation Time (min) | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 |

1- P: Parallel arrangement of activities; S: Serial arrangement of activities

* Percentage of deviation of the result from optimal solution

## 4.6 Model Validation

To verify the results of optimizations using IBM ILOG Optimization Studio, the following procedure was implemented for projects containing up to 180 activities:

- After a TCT optimization using IBM ILOG Optimization Studio, the mode selection results were extracted from Optimization Studio and were manually entered into the corresponding Excel model for that specific project. For instance the mode selection results of optimization of the project consisting of 36 activities with parallel arrangement of 18-activity network (36 P), were entered manually into the corresponding spreadsheet model (36 P).

- The results of choosing those specific modes of construction for a project generate values for duration and total cost in the spreadsheet model. These values should be identical to the duration and total cost obtained by optimization process using IBM ILOG Optimization Studio.

Using the same procedure for projects containing up to 180 activities, the identical results proved that the TCT model in OPL language is valid and the results are reliable. The same model, then, were used for projects consisting larger number of activities.

## 4.7 Further Experimentation

The objective function for the experimentations utilizing traditional optimization methods was to reduce the total cost while meeting the deadline of the project. In the experimentations of this chapter up to this point, the same objective function was used. The single-objective

function is useful for comparing the result with the optimal solution and provides the deviation of the result from the optimal solution as a measure of efficiency of the optimization method. However, the objective function can be modified to a multi-objective function to consider minimizing both time and cost simultaneously. In such a case, penalty, incentive, and indirect cost amounts would define the relationship between the competing objectives of time and cost.

**Table 4-3 Optimization Results of the 18-Activity Project for Various Incentive and Penalty Values**

| Experiment Number | Incentive | Penalty | Duration | Total Cost |
|---|---|---|---|---|
| 1 | 0 | 0 | 110 | 216,270 |
| 2 | 1,000 | 2,000 | 110 | 216,270 |
| 3 | 2,000 | 4,000 | 110 | 216,270 |
| 4 | 3,000 | 6,000 | 101 | 202,420 |
| 5 | 4,000 | 8,000 | 101 | 193,420 |
| 6 | 5,000 | 10,000 | 101 | 184,420 |
| 7 | 6,000 | 12,000 | 101 | 175,420 |
| 8 | 7,000 | 14,000 | 101 | 166,420 |
| 9 | 8,000 | 16,000 | 101 | 157,420 |
| 10 | 9,000 | 18,000 | 100 | 148,270 |
| 11 | 10,000 | 20,000 | 100 | 138,270 |

Deadline = 110 days, Indirect Cost = 1000 /day

In the previous experimentations daily indirect cost was $1000, and penalty and incentive were set to zero in order to force the duration to converge to the user-defined deadline.

Defining positive values for penalty and incentive and eliminating the constraint "Duration<=Deadline" in the model file facilitates the optimization of time and cost simultaneously.

To examine the results of the multi-objective function using constraint programming method, some experimentation are performed. Table 4-3 shows the experimentations results of the 18-activity project for various values of incentive and penalty. Table 4-4 and Table 4-5 present the optimization results for the 36-activity project with parallel arrangement of activities and 36-activity project with serial arrangement of activities respectively.

**Table 4-4 Optimization Results of the 36-Activity Project (Parallel Arrangement) for Various Incentive and Penalty Values**

| Experiment Number | Incentive | Penalty | Duration | Total Cost |
|---|---|---|---|---|
| 1 | 0 | 0 | 110 | 322,540 |
| 2 | 5,000 | 10,000 | 110 | 322,540 |
| 3 | 7,000 | 14,000 | 110 | 322,540 |
| 4 | 8,000 | 16,000 | 105 | 320,540 |
| 5 | 9,000 | 18,000 | 105 | 314,540 |
| 6 | 10,000 | 20,000 | 110 | 322,540 |
| 7 | 12,000 | 24,000 | 101 | 249,840 |
| 8 | 15,000 | 30,000 | 101 | 222,840 |
| 9 | 17,000 | 34,000 | 101 | 204,840 |
| 10 | 19,000 | 38,000 | 101 | 186,840 |
| 11 | 20,000 | 40,000 | 101 | 177,840 |

Deadline = 110 days, Indirect Cost = 1000 /day

**Table 4-5 Optimization Results of the 36-Activity Project (Serial Arrangement) for Various Incentive and Penalty Values**

| Experiment Number | Incentive | Penalty | Duration | Total Cost |
|---|---|---|---|---|
| 1 | 0 | 0 | 220 | 432,540 |
| 2 | 1,000 | 2,000 | 220 | 432,540 |
| 3 | 2,000 | 4,000 | 211 | 428,490 |
| 4 | 3,000 | 6,000 | 220 | 432,540 |
| 5 | 4,000 | 8,000 | 202 | 387,640 |
| 6 | 5,000 | 10,000 | 211 | 400,690 |
| 7 | 6,000 | 12,000 | 202 | 351,640 |
| 8 | 8,000 | 16,000 | 202 | 315,640 |
| 9 | 10,000 | 20,000 | 200 | 277,840 |
| 10 | 12,000 | 24,000 | 201 | 240,490 |
| 11 | 15,000 | 30,000 | 201 | 183,490 |

Deadline = 220 days, Indirect Cost = 1000 /day

## 4.8 Discussion of the Results

The results of using constraint programming techniques are substantially better than the results of using mathematical and evolutionary-based methods which were investigated in the previous chapter, both in terms of solution quality and processing speed. While mathematical and genetic algorithms methods are satisfactory methods of solving TCT problems for small size project networks, they are not efficient anymore once the size of the problem increases. In fact, for solving very large TCT problems, constraint programming proved to be much more efficient than the traditional optimization methods.

Assuming the acceptable deviation of the results from the optimal solution to be %15, the results of traditional optimization methods for the projects consisting of only 54 activities are similar to the results of constraint programming method for the projects consisting of 5400 activities. Considering the exponential growth of the solution space ($3^{54} = 5.8x10^{25}$ versus $3^{5400} = 2.8x10^{2576}$) which is $4.9x10^{2550}$ time more for the 5400 activity project comparing to the 54 activity project, reveals that the constraint programming method tremendously performs better than traditional optimization methods for solving TCT problems.

Further experimentations (Section 4.7) show that constraint programming method can also be used for simultaneous optimization of time and cost. However, the values of indirect cost, incentive and penalty play an important role in the results of the multi-objective model. While minimizing total cost, small amounts of indirect cost and insignificant penalty induce the optimization process to select normal modes of construction and, therefore, to prolong the duration of the project, but large amount of indirect cost and significant penalty reduce the duration of the project.

Large incentive value leads the optimization engine to select crashed modes of activities while minimizing cost and, therefore, reduce the total duration of the project. Minor incentive value enables the total duration to be close to the deadline. In optimization of time and cost simultaneously, the assignment of appropriate values to indirect cost, incentive and penalty are very important and changes in any of these values affect the total project duration and cost.

## 4.9 Summary and Conclusions

In this chapter constraint programming was introduced as a novel approach to solve TCT problems, its differences and similarities to mathematical optimization methods were discussed, and the IBM ILOG Optimization Studio was introduced as a leading commercial optimization package for implementing constraint programming techniques. An optimization model was then developed using the Optimization Programming Language (OPL) for the TCT problem and various experiments were performed. The results of applying constraint programming methods using CP Optimizer of IBM ILOG Optimization Studio were then discussed and compared to the results of applying traditional optimization methods which were performed in the previous chapter.

While in optimization of projects with traditional optimization methods the number of activities was limited to a few hundred activities, utilizing constraint programming facilitated the optimization of vary large projects consisting of thousands of activities. As such, constraint programming offers a significant improvement on the optimization of time and cost for large-scale construction projects. Summary of all the experimentations and the results of using mathematical and evolutionary-based methods on TCT problems are presented in Appendix A; the summary for utilizing constraint programming methods are presented in Appendix C.

# Chapter 5
# Conclusions and Future Research

## 5.1 Conclusions

Combinatorial optimization problems are ubiquitous in several different disciplines and have many practical applications (P. Hentenryck et al., 1999). Considering the current fierce and competitive economy, companies continuously try to improve solutions to their computationally difficult problems, such as scheduling, sequencing, corporate planning, resource allocations, etc., in order to optimize their overall performance. Improved solutions to these problems provide them with a competitive advantage in achieving market leadership; likewise, inefficiency in this technological development may contribute to their failure in the long run.

In large construction projects, optimization of time and cost is a vital competitive advantage for construction firms. Time and cost optimization of large-scale construction projects is a combinatorial problem, and computationally difficult to solve. In recent decades, considerable research has been devoted to solve this optimization problem as efficiently as possible. Innovative optimization methods such as genetic algorithms, in addition to improved calculation power of computers due to faster processing speeds and parallel computing, have improved solutions to optimization problems. However, with mega-projects comprising thousands of activities, the size and complexity of the problems have also increased tremendously.

Small combinatorial problems can be solved even optimally using various heuristic or optimization methods. However, in large-scale combinatorial problems, finding the optimal solution may be extremely difficult and time-consuming, if at all possible. Therefore, from a pragmatic point of view, the quality of the solution for large-scale problems can be determined by the amount by which it is close to the optimal solution, if the optimal solution can be determined.

In this research, the complex nature of time and cost optimization in construction projects were investigated, traditional optimization methods for solving TCT problems, such as mathematical programming and evolutionary-based methods were examined and their inefficiency in solving large TCT problems was verified. Constraint programming techniques were, then, employed for solving large TCT problems. Constraint programming proved to be the best available method for solving large scale TCT problems. Some of the findings and conclusions of the research are as follows:

- The initial experimentations using mathematical and evolutionary-based optimization methods demonstrated that a considerable amount of time is required for these methods to find a feasible solution, after which they start to improve the solution. Thus, by initializing the decision variables with the most crashed mode of construction for all the activities, the experimentations initialized with a feasible solution to eliminate the required time of finding the initial feasible solution.

- The OPL model using constraint programming method, on the other hand, demonstrated to be significantly fast in finding the initial feasible solution comparing to traditional

optimization methods. The decision variables, therefore, are not initialized in the OPL model. Finding a feasible solution and improving it was part of the optimization process in the constraint programming experimentations.

- Modeling with OPL language requires spending considerable amount of time to learn and master essential skills of coding in Optimization Programming Language; however, the effort well worth the efficiency, speed and robustness of the model.

- The OPL model is quite flexible in terms of any required modification within the model file, as well as, within the external spreadsheet or database. Since actual data is separated from the model and is stored in and retrieved from a separate spreadsheet, any change in the relationships of activities and modes of construction can straightforwardly be implemented.

- Due to availability of specific keywords in OPL language for modeling of scheduling problems, the model is very compact and requires minimum external data. For example, instead of defining all the relationships (successors and predecessors) of activities for defining a project network, stating only the successors of each activity, is all it requires to construct the network of the project.

- The model can be implemented as an add-on to project management software extracting the required data such as the relationships between activities, in addition to, obtaining user input such as deadline, penalty, incentive, direct cost and indirect cost in order to optimize the project within the project management software.

## 5.2 Contributions

This research contributes considerably to improve the limitations of solving large-scale discrete time-cost trade-off (DTCT) problems. While the solution methods for solving DTCT problems in the literature are limited to a few hundred activities (Kandil & El-Rayes, 2005), using constraint programming techniques, the number of activities increases to thousands of activities without compromising the quality of solution, within an acceptable range of less than 15% deviation from the optimal solution.

The following are a summary of contributions of the research:

- Developed a flexible time-cost trade-off (TCT) model in Microsoft Excel to be used for applying traditional optimization methods such as mathematical and evolutionary-based methods to solve TCT problems.

- Investigated various optimization tools for solving large-scale TCT optimization problems. Suitable tools for modeling and solving TCT problems were chosen and their efficiency in solving large size problems was examined.

- Verified the inefficiency of traditional optimization methods for solving large-scale TCT problems by several experimentations using mathematical and evolutionary-based optimization methods.

- Developed a TCT model using Optimization Programming Language (OPL) within IBM ILOG Optimization Studio to implement constraint programming techniques for solving large-scale TCT optimization problems.

- The developed constraint programming TCT model proved its ability to solve very large-scale TCT problems. The solutions are satisfactory near optimum (mostly with less than 15% deviation from the optimal solution) with an acceptable processing time (less than 30 minutes).

- Compared the results of using constraint programming method with the results of traditional optimization methods and verified the superiority of constraint programming method in terms of the quality of the solution and the optimization speed compared to the traditional optimization method such as mathematical and evolutionary-based methods.

- Concluded on the superiority of the constraint programming technique comparing to the traditional optimization methods which was utilized within IBM ILOG Optimization Studio.

## 5.3 Future Research

In spite of the significant improvements on the time and cost optimization of large-scale construction projects presented in this research, various other enhancements are offered for the future extensions of the current research, including:

- Investigating the performance of other state-of-the-art optimization packages for optimization of time and cost in construction projects which have the constraint programming capabilities such as Microsoft Solver Foundation.

- Extending the number of modes of construction for each activity to more than three modes on construction. Consequently, the model would turn into a more complicated combinatorial optimization problem which would be harder and more time consuming to solve.

- In this research an Excel-based model was used for implementing mathematical programming methods, as well as, evolutionary-based optimization methods for solving TCT problems. Mathematical programming methods can also be applied to models developed by modeling languages such as OPL and GAMS. Although mathematical formulation of DTCT problems in a modeling language is complicated, it can be a worthy attempt to compare the results of non-Excel-based mathematical model with the results of constraint programming model.

- Expanding the optimization model to include resource allocation and resource leveling constraints, in order to perform resource utilization while optimizing time and cost. This would provide a more complete optimization strategy for construction projects.

- Providing an interface to project management software such as Microsoft Project in order to import and export model data to/from project management software directly.

# Appendix A

## Summary of the Results for Mathematical and Evolutionary-Based Methods

| | No. of Activities | 18 | 36 P [1] | 36 S [1] | 54 P | 54 S | 180 P | 180 S |
|---|---|---|---|---|---|---|---|---|
| | Project Deadline | 110 | 110 | 220 | 110 | 330 | 110 | 1100 |
| | Optimum Solution | 216,270 | 322,540 | 432,540 | 428,810 | 648,810 | 1,172,700 | 2,162,700 |
| Excel Solver | Duration (days) | 110 | 110 | 220 | 110 | 330 | | |
| | Total Cost (×1000) | 254,620 | 409,840 | 433,094 | 607,435 | 769,460 | N/A | N/A |
| | Deviation* | 18% | 27% | 0% | 42% | 19% | Too Many Variable Cells | Too Many Variable Cells |
| | Calculation Time (min) | 2 | 5 | 30 | 30 | 13 | | |
| Risk Solver Platform Standard SLGRG Nonlinear | Duration (days) | 110 | 110 | 220 | 110 | 330 | 110 | 1100 |
| | Total Cost (×1000) | 216,270 | 385,890 | 478,540 | 454,198 | 698,851 | 1,336,900 | 2,315,071 |
| | Deviation* | 0% | 20% | 11% | 6% | 8% | 14% | 7% |
| | Calculation Time (min) | 1.5 | 14 | 30 | 26 | 30 | 30 | 30 |
| Risk Solver Platform Large-scale GRG Solver | Duration (days) | 110 | 110 | 220 | 110 | 330 | 110 | 1100 |
| | Total Cost (×1000) | 216,270 | 379,990 | 513,369 | 454,198 | 714,551 | 1,583,095 | 2,764,677 |
| | Deviation* | 0% | 18% | 19% | 6% | 10% | 35% | 28% |
| | Calculation Time (min) | 1.5 | 18 | 30 | 26 | 30 | 30 | 30 |

# Appendix A Continued:

| | No. of Activities | 18 | 36 P [1] | 36 S [1] | 54 P | 54 S | 180 P | 180 S |
|---|---|---|---|---|---|---|---|---|
| | Project Deadline | 110 | 110 | 220 | 110 | 330 | 110 | 1100 |
| | Optimum Solution | 216,270 | 322,540 | 432,540 | 428,810 | 648,810 | 1,172,700 | 2,162,700 |
| Risk Solver Platform Standard Evolutionary Solver | Duration (days) | 110 | 110 | 219 | 110 | 310 | 110 | 1010 |
| | Total Cost (×1000) | 275,320 | 438,440 | 566,640 | 618,260 | 1,018,260 | 1,807,000 | 3,607,000 |
| | Deviation* | 27% | 36% | 31% | 44% | 57% | 54% | 67% |
| | Calculation Time (min) | 18 | 15 | 1 | 1 | 1 | 21 | 30 |
| Evolver | Duration (days) | 110 | 109 | 220 | 110 | 330 | 109 | 1063 |
| | Total Cost (×1000) | 238,070 | 373,790 | 484,640 | 500,610 | 700,410 | 1,801,700 | 3,087,390 |
| | Deviation* | 10% | 16% | 12% | 17% | 8% | 54% | 43% |
| | Calculation Time (min) | 30 | 30 | 30 | 30 | 30 | 30 | 30 |

1- P: Parallel arrangement of activities; S: Serial arrangement of activities

* Percentage of deviation of the result from optimal solution

# Appendix B

## OPL Model of TCT Problem for IBM ILOG Optimization Studio

```
/*********************************************
 * OPL 12.2 Model
 * Author: Behrooz
 * Creation Date: 2011-08-10 at 7:49:08 PM
 *********************************************/

using CP;

int Deadline = ...;
int Penalty = ...;
int Incentive = ...;
int IndirectCost = ...;

tuple Activity {
  key int id;
  int suc1;
  int suc2;
  int suc3;
}
{Activity} Activities = ...;

tuple Mode {
  key int ActivitiyId;
  key int id;
  int duration;
  float cost;
}
{Mode} Modes = ...;

dvar interval activity[T in Activities];
dvar interval mode[M in Modes] optional size M.duration;

execute {
  cp.param.FailLimit = 1000000;
  cp.param.timeLimit= 1800;
}

minimize (sum (M in Modes) M.cost * presenceOf(mode[M]))* 1000 +
  (max(t in Activities) endOf(activity[t])) * IndirectCost +
            (((max(t in Activities) endOf(activity[t])) - Deadline) >= 0 ?
             ((max(t in Activities) endOf(activity[t])) - Deadline) * Penalty :
             ((max(t in Activities) endOf(activity[t])) - Deadline) * Incentive) ;
```

```
subject to {
 cModes:
 forall (T in Activities)
   alternative(activity[T], all(M in Modes: M.ActivitiyId==T.id) mode[M]);
 cSum:
   max(t in Activities) endOf(activity[t]) <= Deadline;
 cSuccessors:
 forall (Act in Activities){
   if (Act.suc1 != 0)
   endBeforeStart(activity[Act], activity[<Act.suc1>]);
   if (Act.suc2 != 0)
   endBeforeStart(activity[Act], activity[<Act.suc2>]);
   if (Act.suc3 != 0)
   endBeforeStart(activity[Act], activity[<Act.suc3>]);
 }
}

execute {
 for (var M in Modes) {
   if (mode[M].present)
     writeln("Activity " + M.ActivitiyId + " with mode " + M.id +
           " starting at " + mode[M].start + " ending at " + mode[M].end);
 }
 writeln ("Objective value: ", cp.getObjValue());

   }
```

# Appendix C

## Summary of the Results for Constraint Programming Method Using IBM ILOG Optimization Studio

| No. of Activities | 18 | 36 P [1] | 36 S [1] | 54 P | 54 S | 180 P | 180 S |
|---|---|---|---|---|---|---|---|
| Project Deadline | 110 | 110 | 220 | 110 | 330 | 110 | 1100 |
| Optimum Solution | 216,270 | 322,540 | 432,540 | 428,810 | 648,810 | 1,172,700 | 2,162,700 |
| Duration (days) | 110 | 110 | 220 | 110 | 330 | 110 | 1100 |
| Total Cost (×1000) | 216,270 | 322,540 | 432,540 | 428,810 | 648,810 | 1,172,700 | 2,183,700 |
| Deviation* | 0% | 0% | 0% | 0% | 0% | 0% | 1% |
| Calculation Time (min) | 9 | 10 | 10 | 10 | 10 | 13 | 14 |

| No. of Activities | 1800 P | 1800 S | 3600 P | 3600 S | 5400 P | 5400 S | 10800 P | 10800 S |
|---|---|---|---|---|---|---|---|---|
| Project Deadline | 110 | 11000 | 110 | 22000 | 110 | 33000 | 110 | 66000 |
| Optimum Solution | 10,737,000 | 21,627,000 | 21,364,000 | 43,254,000 | 31,991,000 | 64,881,000 | 63,872,000 | 129,762,000 |
| Duration (days) | 110 | 11000 | 110 | 22000 | 110 | 33000 | 110 | 66000 |
| Total Cost (×1000) | 10,737,000 | 23,592,960 | 22,031,750 | 47,121,800 | 33,593,600 | 73,272,560 | 82,352,000 | 146,615,840 |
| Deviation* | 0% | 9% | 3% | 9% | 5% | 13% | 29% | 13% |
| Calculation Time (min) | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 |

# References


Afshar, A., Ziaraty, A. K., Kaveh, A., & Sharifi, F. (2009). Nondominated Archiving Multicolony Ant Algorithm in Time–Cost Trade-Off Optimization. *Journal of Construction Engineering and Management*, *135*(7), 668. doi:10.1061/(ASCE)0733-9364(2009)135:7(668)

Azaron, A., Perkgoz, C., & Sakawa, M. (2005). A genetic algorithm approach for the time-cost trade-off in PERT networks. *Applied Mathematics and Computation*, *168*(2), 1317–1339. doi:10.1016/j.amc.2004.10.021

Bradley, S. P., Hax, A. C., & Magnanti, T. L. (1977). *Applied Mathematical Programming*. Addison-Wesley. Retrieved from http://web.mit.edu/15.053/www/

Burns, S. A., Liu, L., & Feng, C.-W. (1996). The LP/IP hybrid method for construction time-cost trade-off analysis. *Construction Management and Economics*, *14*(3), 265–276.

Castro-Lacouture, D., Süer, G. A., Gonzalez-Joaqui, J., & Yates, J. K. (2009). Construction project scheduling with time, cost, and material restrictions using fuzzy mathematical models and critical path method. *Journal of Construction Engineering and Management*, *135*(10), 1096–1104.

Charnes, A., Cooper, W. W., & Thompson, G. L. (1964). Critical Path Analyses via Chance Constrained and Stochastic Programming. *Operations Research*, *12*(3), 460–470.

Chen, S.-P., & Tsai, M.-J. (2011). Time-cost trade-off analysis of project networks in fuzzy environments. *European Journal of Operational Research*, *212*(2), 386–397.

De, P., Dunne, E. J., Ghosh, J. B., & Wells, C. E. (1997). Complexity of the discrete time-cost tradeoff problem for project networks. *Operations Research*, *45*(2), 302–306.

De, P., James Dunne, E., Ghosh, J. B., & Wells, C. E. (1995). The discrete time-cost tradeoff problem revisited. *European Journal of Operational Research*, *81*(2), 225–238.

Dechter, R. (2003). *Constraint processing*. Morgan Kaufmann.

Elbeltagi, E., Hegazy, T., & Grierson, D. (2005). Comparison among five evolutionary-based optimization algorithms. *Advanced Engineering Informatics*, *19*(1), 43–53.

Elmaghraby, S. E. (1993). Resource allocation via dynamic programming in activity networks. *European Journal of Operational Research*, *64*(2), 199–215. doi:10.1016/0377-2217(93)90177-O

El-Rayes, K., & Kandil, A. (2005). Time-cost-quality trade-off analysis for highway construction. *Journal of Construction Engineering and Management*, *131*(4), 477–486.

Feng, C.-W., Liu, L., & Burns, S. A. (1997). Using genetic algorithms to solve construction time-cost trade-off problems. *Journal of Computing in Civil Engineering*, *11*(3), 184–189.

Feng, Chung-Wei, Liu, L., & Burns, S. A. (2000). Stochastic Construction Time-Cost Trade-Off Analysis. *Journal of Computing in Civil Engineering*, *14*(2), 117. doi:10.1061/(ASCE)0887-3801(2000)14:2(117)

Fondahl, J. W. (1962). *A non-computer approach to the critical path method for construction industry.* Stanford: Dept. of Civil Engineering, Stanford University.

*Frontline Solvers User Guide*. (2011). (V11 ed.). Frontline Systems, Inc.

Gassmann, H. I., & Ireland, A. M. (1995). Scenario formulation in an algebraic modelling language. *Annals of Operations Research*, *59*(1), 45–75.

Golenko-Ginzburg, D., & Gonik, A. (1997). Stochastic network project scheduling with non-consumable limited resources. *International Journal of Production Economics*, *48*(1), 29–37. doi:10.1016/S0925-5273(96)00019-9

*Guide to Using Evolver, The Genetic Algorithm Solver for Microsoft Excel*. (2010). (V5.7 ed.). Palisade Corporation.

Gutjahr, W. J., Strauss, C., & Wagner, E. (2000). A Stochastic Branch-and-Bound Approach to Activity Crashing in Project Management. *INFORMS J. on Computing*, *12*(2), 125–135. doi:10.1287/ijoc.12.2.125.11894

Hegazy, T. (1999). Optimization of construction time - Cost trade-off analysis using genetic algorithms. *Canadian Journal of Civil Engineering*, *26*(6), 685–697.

Hegazy, T., & Ayed, A. (1999). Simplified spreadsheet solutions: Models for critical path method and time-cost-tradeoff analysis. *Cost Engineering (Morgantown, West Virginia)*, *41*(7). Retrieved from http://www.scopus.com/inward/record.url?eid=2-s2.0-0345580817&partnerID=40&md5=1ea6fd4c6a7931c7b21093f84a27345f

Hegazy, Tarek, & Menesi, W. (2010). Critical Path Segments Scheduling Technique. *Journal of Construction Engineering and Management*, *136*(10), 1078. doi:10.1061/(ASCE)CO.1943-7862.0000212

Hegazy, Tarek. (2001). *Computer-Based Construction Project Management* (1st ed.). Prentice Hall.

Hendrickson, C., & Au, T. (1989). *Project management for construction: fundamental concepts for owners, engineers, architects, and builders*. Chris Hendrickson.

Hentenryck, P., Michel, L., Laborie, P., Nuijten, W., & Rogerie, J. (n.d.). Combinatorial Optimization in OPL Studio. In P. Barahona & J. J. Alferes (Eds.), *Progress in Artificial Intelligence* (Vol. 1695, pp. 1–15). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from http://www.springerlink.com/content/pyub8kt9u329xgg5/

Hentenryck, P., Michel, L., Perron, L., & Régin, J.-C. (1999). Constraint Programming in OPL. In G. Nadathur (Ed.), *Principles and Practice of Declarative Programming* (Vol. 1702, pp. 98–116). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from http://www.springerlink.com/content/w7v4135554847126/

Hentenryck, P. V. (1999). *The OPL Optimization Programming Language*. The MIT Press.

Hentenryck, Pascal, & Michel, L. (n.d.). OPL Script: Composing and Controlling Models. In K. R. Apt, E. Monfroy, A. C. Kakas, & F. Rossi (Eds.), *New Trends in Constraints* (Vol. 1865, pp. 75–90). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from http://www.springerlink.com/content/p7c1c1vbbx1lqbtv/

*IBM ILOG Optimization Studio V12.2 - OPL Language Reference Manual*. (2010). International Business Machines Corporation.

*IBM ILOG Optimization Studio V12.2 - OPL Language User's Manual*. (2010). International Business Machines Corporation.

Kalvelagen, E. (2009, October 16). Modeling with Excel+OML, A Practical Guide. Amsterdam Optimization Modeling Group LLC.

Kandil, A., & El-Rayes, K. (2005). Parallel computing framework for optimizing construction planning in large-scale projects. *Journal of Computing in Civil Engineering*, *19*(3), 304–312.

Kandil, A., & El-Rayes, K. (2006). Parallel genetic algorithms for optimizing resource utilization in large-scale construction projects. *Journal of Construction Engineering and Management*, *132*(5), 491–498.

Ke, H., Ma, W., & Ni, Y. (2009). Optimization models and a GA-based algorithm for stochastic time-cost trade-off problem. *Applied Mathematics and Computation*, *215*(1), 308–313. doi:10.1016/j.amc.2009.05.004

Kelley, J. E. (1961). Critical-Path Planning and Scheduling: Mathematical Basis. *Operations Research*, *9*(3), 296–320.

Li, H., & Love, P. (1997). Using Improved Genetic Algorithms to Facilitate Time-Cost Optimization. *Journal of Construction Engineering and Management*, *123*(3), 233. doi:10.1061/(ASCE)0733-9364(1997)123:3(233)

Liang, L., Burns, S. A., & Chung-Wei, F. (1995). Construction time-cost trade-off analysis using LP/IP hybrid method. *Journal of Construction Engineering and Management*, *121*(4), 446–454.

Liao, T. W., Egbelu, P. J., Sarker, B. R., & Leu, S. S. (2011). Metaheuristics for project and construction management - A state-of-the-art review. *Automation in Construction*, *20*(5), 491–505.

Meyer, W. L., & Shaffer, L. R. (1963). *Extensions of the critical path method through the application of integer programming, (Book, 1963) [WorldCat.org]*. Retrieved from http://www.worldcat.org/wcpa/ow/287853

Mon, D.-L., Cheng, C.-H., & Lu, H.-C. (1995). Application of fuzzy distributions on project management. *Fuzzy Sets and Systems*, *73*(2), 227–234. doi:10.1016/0165-0114(94)00309-U

Moselhi, O. (1993). Schedule compression using the direct stiffness method. *Canadian Journal of Civil Engineering*, *20*(1), 65–72. doi:10.1139/l93-007

Ng, S. T., & Zhang, Y. (2008). Optimizing construction time and cost using ant colony optimization approach. *Journal of Construction Engineering and Management*, *134*(9), 721–728.

Pagnoni, A. (1990). *Project engineering: computer-oriented planning and operational decision making*. Springer-Verlag.

Patterson, J. H., & Huber, W. D. (1974). A Horizon-Varying, Zero-One Approach to Project Scheduling. *Management Science*, *20*(6), 990–998.

Phillips, S., & Dessouky, M. I. (1977). Solving the Project Time/Cost Tradeoff Problem Using the Minimal Cut Concept. *Management Science*, *24*(4), 393–400.

Pinedo, M. (2009). *Planning and Scheduling in Manufacturing and Services*. Springer.

Prager, W. (1963). A Structural Method of Computing Project Cost Polygons. *Management Science*, *9*(3), 394–404.

Robinson, D. R. (1975). A Dynamic Programming Solution to Cost-Time Tradeoff for CPM. *Management Science*, *22*(2), 158–166.

Rossi, F., Beek, P. van, & Walsh, T. (2006). *Handbook of Constraint Programming* (1st ed.). Elsevier Science.

Schreyer, A. (2011). GA Optimization for MS Excel. Retrieved from http://www.alexschreyer.net/projects/xloptim/

Siemens, N. (1971). A simple CPM time-cost tradeoff algorithm. *Management Science*, 354–363.

Talbot, F. B. (1982). Resource-Constrained Project Scheduling with Time-Resource Tradeoffs: The Nonpreemptive Case. *Management Science*, *28*(10), 1197–1210.

Van Hentenryck, P., & Michel, L. (2009). *Constraint-based local search*.

Vanhoucke, M., & Debels, D. (2007). The discrete time/cost trade-off problem: Extensions and heuristic procedures. *Journal of Scheduling*, *10*(4-5), 311–326.

Xiong, Y., & Kuang, Y. (2008). Applying an Ant Colony Optimization Algorithm-Based Multiobjective Approach for Time–Cost Trade-Off. *Journal of Construction Engineering and Management*, *134*(2), 153. doi:10.1061/(ASCE)0733-9364(2008)134:2(153)

Zheng, D. X. M., Ng, S. T., & Kumaraswamy, M. M. (2004). Applying a Genetic Algorithm-Based Multiobjective Approach for Time-Cost Optimization. *Journal of Construction Engineering and Management*, *130*(2), 168. doi:10.1061/(ASCE)0733-9364(2004)130:2(168)

Zheng, D. X. M., Ng, S. T., & Kumaraswamy, M. M. (2005). Applying Pareto Ranking and Niche Formation to Genetic Algorithm-Based Multiobjective Time-Cost Optimization. *Journal of Construction Engineering and Management*, *131*(1), 81. doi:10.1061/(ASCE)0733-9364(2005)131:1(81)