HEURISTIC AND EXACT METHODS FOR THE LARGE-SCALE DISCRETE TIME-COST TRADE-OFF PROBLEMS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

SAMAN AMINBAKHSH

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
CIVIL ENGINEERING

MAY 2018

Approval of the thesis:

HEURISTIC AND EXACT METHODS FOR THE LARGE-SCALE DISCRETE TIME-COST TRADE-OFF PROBLEMS

submitted by **SAMAN AMINBAKHSH** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Civil Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalipçilar _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. İsmail Özgür Yaman _____
Head of Department, **Civil Engineering**

Prof. Dr. Rifat Sönmez _____
Supervisor, **Civil Engineering Dept., METU**

Assoc. Prof. Dr. S. Tankut Atan _____
Co-Supervisor, **Industrial Engineering Dept., Işık University**

Examining Committee Members:

Prof. Dr. M. Talat Birgönül _____
Civil Engineering Dept., METU

Prof. Dr. Rifat Sönmez _____
Civil Engineering Dept., METU

Assoc. Prof. Dr. Selçuk Kürşat İşleyen _____
Industrial Engineering Dept., Gazi University

Asst. Prof. Dr. Aslı Akçamete Güngör _____
Civil Engineering Dept., METU

Asst. Prof. Dr. Saeid Kazemzadeh Azad _____
Civil Engineering Dept., Atılım University

Date: 24.05.2018

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: Saman Aminbakhsh

Signature :

ABSTRACT

HEURISTIC AND EXACT METHODS FOR THE LARGE-SCALE DISCRETE TIME-COST TRADE-OFF PROBLEMS

Aminbakhsh, Saman
Ph.D., Department of Civil Engineering
Supervisor: Prof. Dr. Rifat Sönmez
Co-Supervisor: Assoc. Prof. S. Tankut Atan

May 2018, 231 Pages

Construction industry necessitates formulating impeccable plans by decision makers for securing optimal outcomes. Managers often face the challenge of compromising between diverse and usually conflicting objectives. Particularly, accurate decisions on the time and cost must be made in every construction project since project success is chiefly related to these objectives. This is realized by addressing the time-cost trade-off problem (TCTP) which is an optimization problem and its objective is to identify the set of time-cost alternatives that provide the optimal schedule(s). Due to discreteness of many resources in realistic projects, discrete version of this problem (DTCTP) is of great practical relevance. The Pareto front extension of DTCTP is a multi-objective optimization problem that facilities preference articulation of decision makers by providing them with a set of mutually non-dominated solutions of same quality. Due to the complex nature of DTCTP, the literature on large-scale problems is virtually void; besides, most of the existing methods do not suit actual practices and popular commercial planning software lack tools for solution of DTCTP.

The main focus of this thesis relates to providing means for optimization of real-life-scale Pareto oriented DTCTPs and it aims to contribute to both researchers and practitioners by tightening the gap between the literature and the real-world requirements of the projects. The results of the comparative studies reveal that the proposed methods are successful for solving large-scale DTCTPs and provide the management with a quantitative basis for decisions on selection of the proper alternatives for the real-life-scale construction projects.

Keywords: Discrete Time-Cost Trade-Off Problem, Heuristic, Mixed-Integer Linear Programming, Pareto Front, Particle Swarm Optimization

ÖZ

BÜYÜK ÖLÇEKLİ KESİKLİ ZAMAN-MALİYET ÖDÜNLEŞİM PROBLEMLERİ İÇİN SEZGİSEL VE KESİN YÖNTEMLER

Aminbakhsh, Saman
Doktora, İnşaat Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Rifat Sönmez
Ortak Tez Yöneticisi: Doç. Dr. S. Tankut Atan

Mayıs 2018, 231 Sayfa

İnşaat sektöründeki artan rekabet koşulları, verimli ve başarılı sonuçların sağlaması için karar vericilerin kusursuz planları oluşturmasını gerekli kılmaktadır. Proje yöneticileri, proje hedeflerini sağlamak doğrultusunda çelişkili olabilen hedefler arasında tercih yapmanın zorluğu ile yüzleşebilmektedir. Özellikle, proje başarısı yüksek orantıda bu hedeflere bağlı olduğundan dolayı, her inşaat projesinde zaman ve maliyet konusunda doğru kararlar alınması gerekmektedir. Bir optimizasyon problemi olan zaman-maliyet ödünleşim problemi (ZMÖP)'nin amacı optimal proje program(lar)'ı sağlayan zaman-maliyet alternatiflerini bulmaktır. Yapım projelerinde birçok kaynağın kesikli olması nedeni ile bu problemin kesikli versiyonu (KZMÖP) pratik açıdan büyük önem taşımaktadır. Çok amaçlı optimizasyon problemi olan KZMÖP Pareto eğrisi, aynı kalitede olan ve domine edilmeyen bir dizi çözüm sunarak karar vericilerin tercihlerinin belirlenmesine olanak sağlamaktadır. KZMÖP'nin karmaşık yapısından dolayı, literatürde büyük ölçekli problemler ile ilgili önemli boşluklar bulunmaktadır; ayrıca, mevcut yöntemlerin birçoğu büyük ölçekli uygulamalar için uygun değildir ve yaygın

olarak kullanılan paket planlama programları KZMÖP'n çözümüne yönelik hiçbir alternatif sunmamaktadır.

Bu tezin temel odak noktası gerçek hayat ölçeğindeki KZMÖP'lerin çözümü için Pareto optimizasyonuna yönelik yöntemler geliştirmek, ve literatür ile inşaat projeleri gereksinimleri arasındaki boşluğu azaltarak araştırmacılara ve uygulamacılara katkıda bulunmaktır. Karşılaştırmalı sonuçlar, bu tezde önerilen yöntemlerin büyük ölçekli KZMÖP'lerin çözümünde başarılı olduklarını göstermektedir. Geliştirilen yöntemler proje yöneticilerine büyük ölçekli projelerde uygun alternatiflerin seçimine ilişkin niceliksel bir temel yöntem sağlamaktadır.

Anahtar Kelimeler: Karışık Doğrusal Tamsayılı Programlama, Kesitli Zaman-Maliyet Ödünleşim Problemi, Sezgisel, Kuş Sürüsü Algoritması, Pareto Eğrisi

In ever loving memory of my dear father

In dedication to my mother and my brother with love and eternal appreciation

ACKNOWLEDGEMENTS

Moments of my Ph.D. journey have been shared with several special people without the support and encouragement of whom the completion of this thesis would not have been possible. Although it is impossible to thank every single person involved in this journey, I would like to take this opportunity to show my gratitude to those who have assisted me in a myriad of ways and to whom I am greatly indebted.

It has been a great privilege to spend several years in the civil engineering department of the Middle East Technical University; members of which will always remain dear to me. I would first like to express my heartfelt thanks to my supervisor, Prof. Dr. Rifat Sönmez. A more supportive and considerate supervisor I could not have asked for. His willingness to offer me so much of his time and intellect is the major reason this thesis was completed; thus, I attribute the level of my Ph.D. degree to his invaluable encouragement and guidance. Moreover, I would like to extend my sincere thanks to my co-supervisor, Assoc. Prof. Dr. S. Tankut Atan, for his continuous support and valuable comments. I would also like to pay my deepest gratitude to my thesis monitoring committee members, Prof. Dr. M. Talat Birgönül and Assoc. Prof. Dr. Selçuk Kürşat İşleyen, who made my research successful and assisted me at every point to cherish my goal. Also, I would like to present my special thanks to the examining committee members including Asst. Prof. Dr. Aslı Akçamete Güngör and Asst. Prof. Dr. Saeid Kazemzadeh Azad for their valuable feedbacks.

This thesis study was partially funded by a grant from the Scientific and Technological Research Council of Turkey (TÜBİTAK), Grant No. 213M253. TÜBİTAK's support is gratefully acknowledged.

Words cannot express how grateful I am to my family for their relentless encouragement, support, and patience during the years it took me to finish my Ph.D. degree. I salute them all for the selfless love, care, pain and sacrifice they did to shape my life. This thesis is heartily dedicated to my late father, Prof. Dr. Mohammad Aminbakhsh, a brilliant scholar, a noble person, and a wonderful role model for me who took the lead to heaven before the completion of this thesis. To my dearest darling mother, Dr. Simin Nahaei, who always showed me the value of education and discipline and to whom I owe my life for her constant love and blessings. It is impossible to put into words everything I appreciate about her and to acknowledge all the sacrifices that she has made on my behalf. To my gorgeous friend and to whom I randomly call my brother, Sina Aminbakhsh. Words do not suffice to express all the wisdom, love, and support my one and only brother has given me at every stage of my life. I take pride in acknowledging his unconditional and continuous support, both spiritually and materially, that has always been a great source of motivation. To my angelic late grandmother whose thoughts for me have resulted in this achievement and without her affection, loving upbringing, and nurturing my dreams of excelling in education would have remained mere dreams. My acknowledgement would be incomplete without thanking my uncles, Dr. Mehrdad Nahaei and Prof. Dr. Mohammadreza Nahaei, who have kept me going on my path to success in whatever manner possible and for ensuring that good times kept flowing.

I take this opportunity to show my greatest appreciation and to offer my sincerest gratitude with all my heart to Dr. Leili Nabdel for helping me get through the difficult times, for all the fun times and for all the crazy and good memories. I can barely find the words to thank her for all the emotional support, camaraderie, and caring she provided. I am also especially grateful to two special ladies, Dr. Haleh Mortazavi and Dr. Sahra Shakouri; they have cherished with me every great moment and supported me whenever I needed it.

I have great pleasure in acknowledging my gratitude to my wonderful colleagues and fellow research scholars at METU, Ali Can Tatar, Dr. Babak Rahmani, Emad Rezvankhah, Dr. Mahdi Abbasi-Iranagh, Mert Bilir, Dr. Saeed Kamali, and Dr. Sahra Mohammadi, in propelling me on the course of this thesis and for sharing their experiences.

Last but not least, I would like to express my thankfulness to my dearest friends, Dr. Amir Fadaei, Dr. Arsham Atashi, Dr. Mahdi Mahyar, and Shima Ebrahimi, for their precious friendship, and for all the academic and non-academic supports they provided through all these years. I would also like to show my greatest appreciation to the members of our so-called Joint-Venture, Dr. Handan Gündoğan, Dr. Murat Ayhan, and Özgür Dedekargınoğlu, who incented me to strive towards my goal.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

ACO	Ant Colony Optimization
Acts.	Activities
AHP	Analytic Hierarchy Process
AoA	Activity on Arrow
AoN	Activity on Node
APD	Average Percent Deviation
Avg.	Average
B&B	Branch and Bound
BD	Benders Decomposition
BT	Budget
CA	Cash Availability
CDiff	Cost Difference
CPM	Critical Path Method
CPU	Central Processing Unit
CS	Cost-Slope
CS-H	Cost-Slope Heuristic
DDiff	Duration Difference
DDR3	Double Data Rate type Three
DL	Deadline
DP	Dynamic Programming
DPSO	Discrete Particle Swarm Optimization
DTCTP	Discrete Time-Cost Trade-Off Problem
Dur.	Duration
EM	Electromagnetic
ER	Error Ratio

GA	Genetic Algorithm
GD	Generational Distance
GHz	Gigahertz
HR	Hyperarea Ratio
HS	Harmony Search
HV	Hypervolume
IC	Indirect Cost
ID	Identity
IGD	Inverted Generational Distance
IP	Integer Programming
KZMOP	Kesikli Zaman-Maliyet Ödünleşim Problemi
LP	Linear Programming
MA	Memetic Algorithm
MAWA	Modified Adaptive Weight Approach
Max.	Maximum
MHz	Megahertz
Min.	Minimum
MIP	Mixed-Integer Programming
MILP	Mixed-Integer Linear Programming
NP-Hard	Non-deterministic Polynomial-time Hard
NPV	Net Present Value
ONVG	Overall Non-dominated Vector Generation
PERT	Program Evaluation and Review Technique
PF	Pareto Front
PFPSO	Pareto Front Particle Swarm Optimizer
PM	Parallel Merge
Pred.	Predecessor
PSO	Particle Swarm Optimization
QSA	Quantum Simulated Annealing
RAM	Random Access Memory

RV	Range Variance
SA	Simulated Annealing
SAM	Siemens Approximation Method
SFL	Shuffled Frog Leaping
SM	Serial Merge
Sol.	Solution
St.	Stochastic
Succ.	Successor
TCT	Time-Cost Trade-Off
TCTP	Time-Cost Trade-Off Problem
TP	True Pareto
UCS	Uncrash Cost-Slope
UD	Uniform Distribution
UF	Unified Front
UFF	Uncrash Free-Float
UI	User Interface
USD	United States Dollar
VBA	Visual Basic for Applications
ZMÖP	Zaman-Maliyet Ödünleşim Problemi

CHAPTER 1

INTRODUCTION

Construction industry necessitates formulating impeccable plans by decision makers and construction planners for securing optimal outcomes. Managers often face the challenge of compromising between diverse and usually conflicting objectives of each project. Particularly, accurate decisions on the time, cost, quality, and resource utilization of a project are essential prerequisites of an exhaustive plan. Focusing on the noted aspects coupled with consideration of other impartible components of prosperous project deliveries – such as provision of safety and productivity upkeeps – further narrow the field for the project management team. Of the specified aspects, time and cost are regarded as two of the most significant, yet counteracting factors that need to be considered in every construction project since project success is chiefly related to these objectives. A key process for achieving the anticipated resolutions is preparation of a schedule. Logical relationships – known as the precedence constraints, lag times, working calendars, resource requirements, and contingency plans are some of the many essential concerns for preparing a decent schedule. Idle labor and equipment, delays, dissatisfied customers, disputes, bad reputation are some of the adverse outcomes that might arise in the absence of an adequate schedule. More importantly, in capital intensive construction projects, major financial losses might occur in the light of a suboptimal schedule, or because of small deviations from an optimal solution to the scheduling problem. In contrast, a flawless schedule can obviate occurrence of such problems which ultimately results in completion of a project on or ahead of time.

In project scheduling, critical path method (CPM) is used to determine the completion time of a project by calculating the longest sequence of the activities in the project network which is known as the critical path. Critical path plays a crucial role in planning of a project since any delay in realization of an activity on this path results in overall project delay. Classical network analyses like CPM merely incorporates the time aspect of the projects. Such methods attempt to minimize duration of the project without taking into account the availability of resources (both nonrenewable and physical resources). Generally, it is desirable for the involved parties to minimize the duration in order to finish a construction project ahead of a prescribed completion deadline. Delay penalties can be avoided by finishing a project earlier; besides, managers accelerate a schedule due to many factors like improving cash-flow, avoiding unfavorable climate conditions, early commissioning, earning early-completion incentive, starting another project earlier, and mainly in the interest of increasing profit margins. Known as crashing, any reduction in project duration is facilitated by compression or acceleration of the project schedule. Decision makers speed up the project by optimally crashing selected critical activities that levy least additional cost. Crashing a project schedule is usually facilitated by provisioning resource overloads – i.e. allocating additional manpower and machinery resources or recruiting subcontractors – or by implementing alternative speedy construction techniques. Although shortening the project duration below its normal level enables reducing indirect cost and avoiding potential delay penalties, all the schedule expedition techniques add to the total cost of a project. Obviously, it is because of the fact that the aforementioned crashing approaches require greater financial expenditures known as the direct costs. This trade-off between time and non-renewable resources (e.g., money) of the project is known as the time-cost trade-off problem which is abridged as TCTP and is one of the most important and applicable research areas in project management; especially because of the prevailing emphasis on time-based completion of the construction firms. The objective of general time-cost trade-off problem is to identify the set of time-cost modes (alternatives) that will provide the optimal schedule under certain

conditions. It is a problem solving and decision-making science which provides the management with a quantitative basis for decisions on selection of the proper alternatives. The importance of TCTP has been recognized since development of the CPM (De, Dunne, Ghosh, and Wells, 1995). Starting since early 1960's, several different researches have been conducted to address this problem in the literature.

In the literature, three types of TCTP have been commonly studied; the deadline problem, the budget problem, and the Pareto front problem. The objective of the deadline problem is to determine the set of time-cost alternatives that will minimize the total cost – including direct and indirect costs, penalties and bonuses – for a given project deadline. The budget problem aims to identify the time-cost alternatives to minimize the project duration without exceeding the budget. The Pareto front problem which is also the chief focus of this thesis, is a multi-objective optimization problem and involves determination of the non-dominated time-cost profile over the set of feasible project durations to generate Pareto fronts of the problems (Vanhoucke and Debels, 2007). Pareto front optimization of TCT problems is recognized to be the ultimate resolution of TCTP analyses (e.g., Zheng, Ng, and Kumaraswamy, 2005; Yang, 2007b; Eshtehardian, Afshar, and Abbasnia, 2008; Aminbakhsh and Sonmez, 2017). The importance of Pareto front is emphasized since preferences of decision makers can be articulated by providing them with a set of mutually non-dominated solutions of same quality instead of a single optimal solution. This way decision makers can choose the best solutions based on their own concerns.

Over the years, numerous studies have been conducted to model the time-cost relationship of the project. Early research on TCTP assumed the relation between time and cost to be continues (Kelley and Walker, 1959; Fulkerson, 1961, Siemens, 1971; Goyal, 1975). In recent years there has been increased attention toward the discrete version of the problem due to its great practical relevance. This consideration is imperative to TCT analyses since in practice many resources (e.g.,

workforce, equipment) are available in discrete units; in addition, time-cost function of any type can be estimated by discrete functions. Being the main focus of this thesis, the discrete version of TCTP considers discrete sets of time-cost options for the activities and it is known as the discrete time-cost trade-off problem – abridged as DTCTP – in the literature.

All of the three extensions of DTCTP that were mentioned earlier are Non-deterministic polynomial-time hard (NP-hard) problems in the strong sense (De, Dunne, Ghosh, and Wells, 1997). That is to say, solution of DTCTP requires concurrent searches over the solution space and that any escalation in the size of the project (any growth in the number of activities, modes, or both simultaneously) contributes to the significantly higher computational burden. Due to this fact, exhaustive enumeration is incapable of providing an efficient and convenient mean for DTCT analyses; therefore, researchers have come up with numerous optimization techniques for solution of DTCT problems. The methods proposed for DTCTP could be categorized into three areas: exact methods, heuristics, and meta-heuristics. Traditionally, solution of DTCTP has been modelled by mathematical programming – known as the exact methods – such as linear programming (Kelley, 1961), dynamic programming (Butcher, 1967), hybrid LP/IP programming (Liu, Burns, and Feng, 1995), and branch-and-bound algorithm (Demeulemeester, De Reyck, Foubert, Herroelen, and Vanhoucke, 1998). The literature on heuristics for DTCTP is limited to the methods proposed by Fondahl (1961), Siemens (1971), Goyal (1975), Moselhi (1993), and Bettemir and Birgonul (2017). Evolutionary algorithms are among the meta-heuristics practiced in solution of DTCT problems. Meta-heuristics include genetic algorithm (GA) (Feng, Liu, and Burns, 1997; Zheng, Ng, and Kumaraswamy, 2004), ant colony optimization (ACO) (Xiong and Kuang, 2008), and particle swarm optimization (PSO) (Yang, 2007b).

Generally, exact methods are incapable of obtaining optimal solutions for large-scale problems efficiently. The proposed exact algorithms, while requiring massive

computational resources, are more difficult to implement and are prone to being stagnated in local optima in non-convex solution spaces (De et al., 1995; Feng et al., 1997; Eshtehardian et al., 2008; Afshar, Ziaraty, Kaveh, and Sharifi, 2009). Due to the alleged drawbacks of the mathematical programming models, researchers have turned their interests toward alternative optimization methods. Of the alternative optimization techniques, existing studies providing heuristic algorithms acknowledge that they are problem dependent and cannot handle large-scale problems efficiently (Siemens, 1971). Most of the present heuristics assume merely linear time-cost functions and they fail to solve the Pareto front problem (Feng et al., 1997; Zheng et al., 2005). In addition, the existing meta-heuristics inability to escape from local optima is observed as their main deficiency (Zheng et al., 2005; Sonmez and Bettemir, 2012). In fact, it is worth mentioning that although a large body of the literature has hitherto been dedicated to development of optimization methods for DTCTP, only some of these methods are used in real-life practices. That is largely resulting from the fact that they do not suit actual practices and that major domain of the literature focus on proving applicability of various optimization models rather than providing means for optimization of real construction projects. Accordingly, it might be perceived that there exists a gap between the theoretical achievements of scholars and practical applications of professionals in the field of construction. To expand on this, majority of the past research have used problems including up to only eighteen activities in computational experiments. Very few of the existing methods can be applied to optimization of real-life construction projects which typically comprise more than 300 activities (Liberatore, Pollack-Johnson, and Smith, 2001). Furthermore, a few methods that are tested for real-life-size large-scale problems, require enormous computation time and resources thanks to the inherent complexity of solving DTCTPs. Last but not least, despite the fact that any scientific decision support tool would have a pivotal role in the decision-making process, none of the commercial scheduling software packages (e.g., Microsoft Project, Primavera) include tools or modules for TCT analyses of the scheduling problems.

As mentioned earlier, DTCTP has been unraveled for relatively small instances and the literature on large-scale DTCTP is virtually void thanks to the complex nature of the problem. It was also stated that the ultimate resolution and the most complex extension of DTCTP is the Pareto front problem. Further, despite the dramatic increase in computing speed of the modern computers, they might prove to be insufficient for practical applications. Accordingly, an efficient optimization model for tackling Pareto oriented optimization of real world DTCT problem is long overdue. Recognizing this, the main focus of this thesis is set as Pareto oriented optimization of large-scale DTCT problems and it aims to contribute to both researchers and practitioners by tightening the gap between the literature and the real-world requirements of the projects. In this respect, new models applicable in real projects are developed which are believed that will suit the actual practices of construction managers.

1.1. Problem Description and Mathematical Model

The objective of general DTCTP is to determine the optimal set of time-cost modes that will minimize the sum of direct and indirect costs by taking into account the liquidated damages and bonuses for a project.

1.1.1. Assumptions

The assumptions of the general DTCTP are as follows:

- There is a linear relation between the indirect costs and the project duration;
- The duration of project activities is a discrete, non-increasing function of the amount of resources assigned to them (Vanhoucke and Debels, 2007);
- An activity can start after all of its predecessors are finished;
- Activities cannot be interrupted. Each activity is executed without interruption from its start time to its finish time.

1.1.2. Mathematical Model

The general DTCTP can be formulated by modifying the formulation of De et al. (1995) to include the indirect costs and the delay penalty as follows:

$$minimize \quad \sum_{j=1}^{S}\sum_{k=1}^{m(j)}(dc_{jk}x_{jk}) + D \times ic + T \times dp \tag{1.1}$$

subject to:

$$\sum_{k=1}^{m(j)} x_{jk} = 1 \quad , \quad \forall\, j = \{1,...,S\} \quad and \quad \forall\, k = \{1,...,m(j)\} \tag{1.2}$$

$$T = D - D_{dl} \tag{1.3}$$

$$\sum_{k=1}^{m(j)} d_{jk}x_{jk} + St_j \le St_l \quad , \quad \forall\, l \in Sc_j \quad and \quad \forall\, j = \{1,...,S\} \tag{1.4}$$

$$D \ge Ft_S \tag{1.5}$$

$$St_0 = 0 \tag{1.6}$$

where; dc_{jk} is the direct cost of mode k for activity j; x_{jk} is a binary (0/1) variable and is set to 1 when activity j is undertaken with mode k; ic is the daily indirect cost; D is the project duration; D_{dl} is the project deadline; T is the delay amount of the project; dp is the daily delay penalty; db is the daily bonus; d_{jk} is the duration of mode k for activity j; St_j and Ft_j are the start time and the finish time for activity j, respectively; and Sc_j is the set of immediate successors for activity j.

Objective function of a general DTCTP is defined as Eq. (1.1) which attempts to minimize the summation of the direct and the indirect costs, i.e., the total cost of the project. The constraint defined as Eq. (1.2) secures selection of only a single time-cost alternative for each of the activities. The project delay is calculated using Eq. (1.3) by subtracting the imposed completion deadline from the total duration. The precedence relationships are maintained by means of Eq. (1.4). Completion time of the project is determined using Eq. (1.5) which ensures total duration not to be smaller than the finish time of the final activity. Eq. (1.6) reflects the start time of the initial activity which is set to be equal to zero.

1.2. Scope and Objectives

1.2.1. Scope

This study concerns development of an exact, a heuristic, and a meta-heuristic for the deadline and Pareto front classes of DTCTP. The main focus of the study is the Pareto front optimization of large-scale projects.

1.2.2. Objectives

The primary objective of this thesis is to design and develop state-of-the-art methods that can achieve successful results for discrete time-cost optimization of large-scale projects. For Pareto oriented optimization of large-scale projects, the proposed method aims to provide a large number of non-dominated solutions with minimal or no deviations from the optimal solutions in a short amount of computation time.

1.3. Research Methodology

New models applicable in real projects are designed and developed to achieve the research objectives. The new proposed models include Mixed-Integer Linear Programming techniques that use Gurobi solver version 6.0.5, new Particle Swarm Optimizers, and new Cost-Slope Heuristics. For all the proposed methods, two variants have been designed and developed to address both the deadline and the Pareto front classes of DTCT problems. Since, exact procedures are the only methods guaranteeing optimality of the solutions and that heuristics and meta-heuristics are incapable of securing the optimality of the solutions, the proposed Mixed-Integer Linear technique is used in performance evaluation of the developed heuristic and meta-heuristic methods. Exact solutions are used to validate the accuracy of the results obtained by the proposed algorithms. The developed Mixed-Integer Linear Programming technique incorporates an efficient Upper-bound calculation for reducing the size of the solution space and a new merging technique is also applied which exponentially decreases the scale of the practiced problems. Resultantly, computation time of DTCTP analyses is significantly reduced in the light of the implemented procedures. The proposed Particle Swarm Optimizer is equipped with a unique semi-deterministic initialization technique and uses new principles for Presentation and Position-updating of the particles. Regarded as the chief contribution of this thesis, the novel Cost-Slope Heuristic engages unique scientific and programmable rules which enjoys the fastness for solving DTCTP. For the proposed Cost-Slope Heuristic, an original method for CPM calculations is designed to accelerate the solutions process. Furthermore, similar to the Mixed-Integer Linear Programming method, a merging technique is implemented to the Cost-Slope Heuristic for the sake of reducing scale and computation cost of the practiced problems.

This thesis study also presents integration of DTCTP optimization modules into Microsoft Project – a widely used commercial planning software in the construction

industry – by means of an add-in which is capable of solving two variants of DTCTP, namely, deadline and Pareto problems. The integrated modules include both the proposed Particle Swarm Optimizer and the Cost-Slope Heuristic. Benefiting from the presented add-in, users of Microsoft Project will readily be able to visualize the optimized schedules for the practiced projects; hence, the proposed methods are supposed to be more readily accepted and used by the parties to construction projects. It is expected that these approaches might prove to be an efficient and effective base for exerting this highly challenging problem.

In order for better evaluation of the capabilities of the proposed optimization models and methods, new sets of DTCT problems have been generated by means of a project instance generator ProGen/Max that includes multi-mode problems with up to 500 activities; in addition, a random network generator RanGen2 is also used to generate new sets of multi-mode DTCTPs including up to 990 activities, with more complex networks and more realistic sets of time-cost alternatives. Along with the systematically generated large-scale instances, benchmark problems – acquired from the literature – and case projects are also simulated and solved using the proposed methods and their performances are evaluated and compared against existing optimization approaches according to a set of performance comparison indices. All the proposed algorithms and performance evaluation procedures have been implemented in C++ and C# programming languages using Microsoft Visual Studio 2013. The proposed approaches engage CPM calculations for logical relationships of type finish-to-start, considering no lags in between. The results of the comparative studies are promising and reveal that the obtained solutions not only are comparable, but also are better than previous approaches. Experiments also attest to the efficacy of the proposed methods for successful solution of real-life-size large-scale DTCT problems within only moderate computational effort.

The sequel of the thesis is structured as follows. Chapter 2 starts with an introduction on CPM and TCTP, followed by a detailed review of the existing

research on TCTP in the construction management domain. Chapter 3 presents the background and theoretical properties of PSO and SAM approaches; furthermore, the details of the proposed DPSO method for cost minimization and deadline extensions of DTCTP, as well as comparative studies on DPSO's performance are described in this chapter. In Chapter 4, followed by an introduction on the Pareto optimality, the proposed PFPSO model for Pareto front extension of DTCTP is explained along with the results of the computational experiments and performance comparisons conducted for this method. Chapter 5 includes the major body of this thesis study, which illustrates new techniques used in development of different variants of CS-Heuristic and MILP methods. This chapter also covers results of the inclusive performance measurements and comparative studies carried out over benchmark problems and new sets of complex instances, for validation of all the proposed approaches. Chapter 6 explains the methodology implemented to develop a Microsoft Project add-in. Chapter 7 includes the concluding remarks and points out some directions for the future research.

CHAPTER 2

LITERATURE REVIEW

In this chapter, the principles of project scheduling are outlined. Insight is given into one of the major classical network analyses, the Critical path method (CPM). Different types of construction project expenses are also described in this chapter. The time-cost trade-off problem (TCTP) is elaborated in addition to the existing solution techniques proposed for this problem. Prospects of exact, heuristic, and meta-heuristic methods for practical solution of large-scale TCT problem are presented.

2.1. Critical Path Method (CPM)

Scheduling is defined by Mubarak (2010) as making judgments on timing and sequence of the work-packages, by means of which, the overall completion date of a project is determined. Preparing schedules for projects is facilitated by conducting network analysis (Lock, 2007). CPM is widely used for network analysis and scheduling of construction projects. This method is capable of calculating the overall completion date of a project by computing the longest path in the project network. In CPM method the longest path in the network show the shortest amount of time required to complete the project (Kerzner, 2009).

CPM calculations take place based on the duration and precedence constraints of the activities. Network illustrations – known as logic diagrams – are used as medium of tracking the critical paths of a project network. In general, the activity

networks of the projects are illustrated either by using the activity on arrow (AoA) or the activity on node (AoN) notation systems. As the names imply, in the first system (AoA) the activities are represented by arrows intersecting nodes which resemble events; while, in the second system (AoN) activities are represented by nodes and the logical relationships are traced by arrows. In this thesis, the activity on node representation system is preferred due to the following reasons:

- They are more flexible and simple to construct;
- Most of the modern project management software support this network chart;
- It is easy to build Gantt-Chart based on the AoN network;
- This representation does not require dummy activities as the arrows represent only the dependencies;
- It is easier to track parallel and serial paths in the network for merging purposes.

In CPM scheduling technique, there might be more than a single critical path of the same length. The remainder of the paths are shorter in length, which contributes to a certain degree of freedom in starting and finishing of the non-critical activities. In contrast, a critical activity without a degree of freedom that its allowable delay is equal to zero, must be started as soon as its predecessors are finished. This degree of freedom, which is hailed as the float or slack time in the project scheduling domain, can be calculated by evaluating the difference between an activity's earliest and latest dates (either start or finish dates). Slack time can be interpreted as the amount an activity can be delayed without affecting the overall completion of the project. Earliest and latest dates of the activities can be determined by implementing forward-pass and backward-pass, respectively.

Consideration of a completion deadline is imperative for construction projects, since, in practice majority of projects include a target date which is stipulated in the contract by the client. Completion deadline of a project is generally the earliest possible date decided by dint of the network analyses such as CPM; nevertheless,

the definite completion date might also be targeted on a later date. The main resolution in time-limited projects is to secure completion of the project on a date, no later than the prescribed deadline. In such projects, any projected resource over-allocation might be dealt with hiring subcontract workforce or by making alternative short-term resource provisions.

In this thesis, due to the practical relevance of the completion deadline in projects, all the systematically generated instances are tackled by assuming a target completion date. In addition, all the proposed methods are equipped with CPM technique for calculation of the early and late dates and floats of the work-packages, as well as the overall completion date of the practiced instances. Original to this thesis, the proposed Cost-Slope Heuristic is also complemented with a unique CPM-esque approach for faster network analyses.

2.2. Time-Cost Trade-off Problem (TCTP)

Of the specified aspects, time and cost are regarded as two of the most significant, yet counteracting factors that need to be considered in every construction project since project success is chiefly related to these objectives in today's market-driven economy. Generally, it is a desirable resolution for the involved parties to minimize the duration in order to finish a construction project ahead of a prescribed completion deadline. Unambiguously, finishing on or under the specified budget is another favorable achievement; hence, simultaneous realization of the noted objectives is sought after undeniably. For this purpose, decision makers evaluate the cost per unit time (cost-Slope) as well as a feasible budget (cash-flow) region for the project. Hence, one of the dominant prospects of the network analysis can be concluded as finding a solution that not only satisfies the completion deadline, but also has the lowest feasible total cost that resides within the feasible budget boundaries. In network analysis, normal duration is defined as the time required to complete the project under ordinary conditions without deliberate delay or

acceleration; further, normal cost is the amount that is required for completion of the project within the normal duration. Despite the fact that it is possible to avoid delay penalties by finishing a project earlier, managers accelerate a schedule due to many other factors like improving cash-flow, avoiding unfavorable climate conditions, early commissioning, earning early-completion incentive, starting another project earlier, and mainly in the interest of increasing profit margins. Usually, the contractor knows the expected mobilization date to another project; in this regard, the contractor might esteem to accelerate the current project in favor of supplying the required resources so that they can be allocated to the new project.

Accelerating a project schedule might be profitable only up to a certain level. This is because of the fact that in practice alternatives of the activities are mutually incomparable with convex relationships; that is, the faster it gets to execute an activity, the more expensive it gets. The act of accelerating (compressing) or crashing a project schedule literally means reducing the duration of a project. Though, there is a fine line between these two approaches. Whilst both the techniques aim at advancing the completion date of a project, accelerating does not necessarily mean targeting to reach the least possible duration. It must be also clarified that the possibility of schedule acceleration or availability of the crashing alternatives for the activities of a project is highly affiliated with its typology. Those with complex precedence constraints usually offer less flexibility regarding the schedule acceleration compared with those with less strict logical relationships.

Decision makers speed up the project by optimally crashing selected critical activities that levy least additional cost. Crashing a project schedule is usually facilitated by provisioning resource overloads – i.e. allocating additional manpower and machinery resources or recruiting subcontractors – or by implementing alternative speedy construction techniques. Although shortening the project duration below its normal level enables reducing indirect cost and avoiding potential delay penalties, all the schedule expedition techniques add to the total cost

of a project. Obviously, it is because of the fact that the aforementioned crashing approaches require greater financial expenditures known as the direct costs.

According to the Construction Industry Institute (1988), acceleration of a project schedule can be facilitated by using more than 90 techniques. Mubarak (2010) has summarized some of the more significant schedule acceleration methods as:

- Reviewing or evaluating the schedule for errors or imperfect precedence relationships;
- Applying fast-tracking;
- Studying constructability and value engineering;
- Assigning over-time schedule or using shift-works;
- Setting incentives for the more productive work-forces;
- Increasing the size of the allocated work-forces;
- Employing more efficient construction techniques;
- Using materials with faster installation processes;
- Enhancing project management and supervision;
- Preventing communications breakdowns.

Since schedule acceleration remarkably influence both the direct and indirect costs of a project, shedding some light on their differences is inevitable and necessary. The main principle for distinguishing the direct expenses from the indirect costs can be depicted as a direct cost item is directly associated with an explicit work item; whereas, the indirect costs cannot be related to any specific task or a particular project. Direct expenses may include labor, material, equipment, subcontractor, machinery, and costs related to fees and permits. On the other hand, indirect costs might include project overhead and general overhead expenditures that contain salaries of the guard, cook, and office personnel as well as the energy costs.

It was already mentioned that in practice, the time-cost relation is convex among the alternatives of an activity. Therefore, the more an activity accelerates, the more the daily cost of acceleration increases. This is because of the fact that over-staffing or working over-time notably decreases the productivity which in turn, increases the ratio of unit cost per unit output. A realistic nonlinear time-direct-cost profile is shown in Figure 2.1.

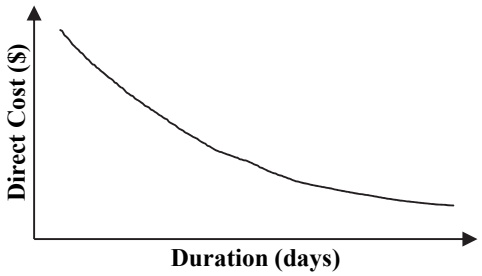


Figure 2.1 – Nonlinear progression of direct cost resulted from schedule acceleration.

The indirect costs, on the other hand, consists of time-dependent and time-independent types. Besides, they usually tend to increase faster at the initial stages of the project and then to remain constant. Nevertheless, they are usually assumed to be linearly comparative to the duration of the project for schedule acceleration purposes. Likewise, for the sake of simplified cost computations, second order cost components (e.g., insurance and bond payments) are usually not included in the daily indirect expenses. As shown in Figure 2.2, the indirect cost is assumed to decrease linearly in case of schedule acceleration.

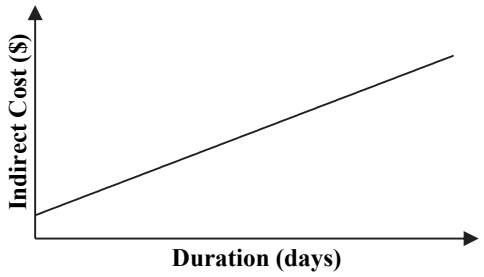


Figure 2.2 – Linear decline of indirect cost resulted from schedule acceleration.

As shown in Figure 2.3, the total cost profile of a project can be obtained by adding up the direct and indirect costs together. Starting from the normal schedule – which consists of least direct cost/largest duration alternatives – the total cost of the project decreases as a result of acceleration till reaching an optimal point; after this point, as the acceleration gets closer to the crash schedule – which consists of largest direct cost/shortest duration alternatives – the total cost surges up at an increasing rate.

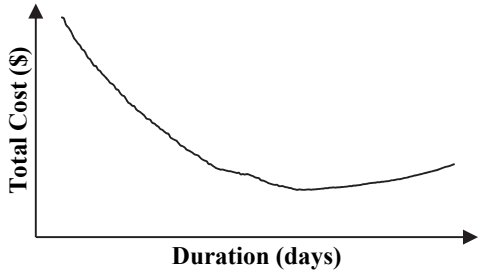


Figure 2.3 – Variation of total cost resulted from schedule acceleration.

This trade-off between time and cost of the project is known as the time-cost trade-off problem which is abridged as TCTP and is one of the most important and applicable research areas in project management. The objective of general time-cost trade-off problem is to identify the set of time-cost modes (alternatives) that will provide the optimal schedule under certain conditions. TCTP mainly attempt to speed up the critical activities while relaxing non-critical ones (Siemens, 1971).

It is a problem solving and decision-making science which provides the management with a quantitative basis for decisions on selection of the proper alternatives. The importance of TCTP has been recognized since development of the CPM (De et al., 1995).

In the literature, three types of TCTP has been commonly studied; the deadline problem, the budget problem, and the Pareto front problem. The objective of the deadline problem is to determine the set of time-cost alternatives that will minimize the total cost – including direct and indirect costs, penalties and bonuses – for a given project deadline. The budget problem aims to identify the time-cost alternatives to minimize the project duration without exceeding the budget. The Pareto front problem which is also the chief focus of this thesis, is a multi-objective optimization problem and involves determination of the non-dominated time-cost profile over the set of feasible project durations to generate Pareto fronts of the problems (Vanhoucke and Debels, 2007). Pareto front optimization of TCT problems is recognized to be the ultimate resolution of TCTP analyses (e.g., Zheng et al., 2005; Yang, 2007b; Eshtehardian et al., 2008; Aminbakhsh and Sonmez, 2017). The importance of Pareto front is emphasized since preferences of decision makers can be articulated by providing them with a set of mutually non-dominated solutions of same quality instead of a single optimal solution. This way decision makers can choose the best solutions based on their own concerns.

Over the years, numerous studies have been conducted to model the time-cost relationship of the project. Early research on TCTP assumed the relation between time and cost to be linear (Kelley and Walker, 1959; Fulkerson, 1961; Siemens, 1971; Goyal, 1975). Subsequently, the assumption of linearity was relaxed allowing for consideration of other types of the objective function, namely, concave function (Falk and Horowitz, 1972), convex function (Foldes and Soumis, 1993), a hybrid of concave and convex functions (Moder, Phillips, and Davis, 1983), and quadratic function (Deckro, Hebert, Verdini, Grimsrud, and Venkateshwar, 1995). In recent

years there has been increased attention toward the discrete version of the problem (Skutella, 1998; Zheng et al., 2004) due to its great practical relevance. This consideration is imperative to TCT analyses since in practice many resources (e.g., workforce, equipment) are available in discrete units; in addition, time-cost function of any type can be estimated by discrete functions. Being the main focus of this thesis, the discrete version of TCTP considers discrete sets of time-cost options for the activities and it is known as the discrete time-cost trade-off problem – abridged as DTCTP – in the literature.

All the three extensions of DTCTP that was mentioned earlier are Non-deterministic polynomial-time hard (NP-hard) problems in the strong sense (De et al., 1997). That is to say, solution of DTCTP requires concurrent searches over the solution space and that any escalation in the size of the project (any growth in the number of activities, modes, or both simultaneously) contributes to the significantly higher computational burden. Moreover, any variation in selection of the alternatives modifies the project schedule which requires rescheduling the project for potential changes in its total cost and total duration amounts. It is obvious that any rescheduling process necessitates reanalyzing the network using the critical path method. Exhaustive enumeration even with the modern computers is, therefore, not a convenient and economically feasible method for solution of DTCTP. Therefore, since the early 1960's – concurrent with introduction of project analysis techniques by Fulkerson (1961) and Kelley (1961) – researchers have come up with several different optimization techniques to address this problem in the literature. Heuristic method of Nicolai Siemens (1971) remains the first notable approach developed for solution of TCTP. The methods proposed for DTCTP can be categorized into three areas: exact methods, heuristics, and meta-heuristics. Sections 2.3, 2.4, and 2.5 present the existing literature on the exact, heuristic, and meta-heuristic methods, respectively.

2.3. Exact methods for TCTP

Traditionally, solution of DTCTP has been modelled by mathematical programming, known as the exact methods. These approaches attempt to explore the entire solution space to find the exact optimal solution. Generally, exact methods are incapable of obtaining optimal solutions for large-scale problems efficiently. The proposed exact algorithms, while requiring massive computational resources, are more difficult to code and implement and are prone to being stagnated in local optima in non-convex solution spaces (De et al., 1995; Feng et al., 1997; Eshtehardian et al., 2008; Afshar et al., 2009). Due to the alleged drawbacks of the mathematical programming models, researchers have turned their focus toward alternative optimization methods. Nevertheless, exact procedures are the only methods guaranteeing optimality of the solutions. Owing to this very reason, they play a crucial role in performance evaluation of alternative optimization methods, viz., heuristics and meta-heuristics. In the absence of an exact solution, the accuracy of the results obtained by other methods cannot be calculated. Some of the most popular variants of the exact algorithms include linear programming, dynamic programming, mixed-integer programming, branch-and-bound algorithm, hybrid LP/IP programming, and Benders decomposition method.

Of the exact methods, mixed-integer programming (MIP) consists of decision variables constrained to include integer values, with objective function and other variables allowed to select non-integer (continuous) values. MIP is a subset of the broader linear programming (LP) in which all the variables and objective functions are linear. TCT problems of any type can be converted into LP/MIP and solved by implementing a commercial optimization solver. LP/MIP approaches suit the nature of real-life TCTPs with convex time-cost relationships. Though, as the number of activities of a project increases, the number of parameters to define for LP/MIP approaches grows significantly.

The MIP approach of Meyer and Shaffer (1965), remains the pioneering attempt for exact solution of TCTP. Another early MIP for TCTP analysis includes the method proposed by Crowston and Thompson (1967). A rather small sample problem with eight activities is used in their study. Liu et al. (1995) proposed a hybrid approach combining linear and integer programming (IP) together. The hybrid approach uses LP to set lower-bounds for cost and uses IP to find the exact solutions. This model is implemented as a macro in Microsoft Excel environment and tested using a small-scale seven activity problem. Moussourakis and Haksever (2004) present a flexible MIP model that tackles deadline and budget TCT problems with linear, piecewise linear, or discrete objective functions. Piecewise linearity of nonlinear continuous functions is the only assumption required for this approach. This method is studied using a seven-activity sample problem. Another MIP is proposed by Chassiakos and Sakellaropoulos (2005) for Pareto front TCT problem, objective function of which includes delay penalty and incentive payments. A small-scale 29-activity sample problem – including finish-to-start with lag/lead time relationship – is fed into the model. A more recent study on MIP is carried out by Szmerekovsky and Venkateshan (2012). This method is capable of handling problems with time-cost alternatives defined as different cost functions. This model is able to minimize Net Present Value (NPV) of costs and maximize Cash Availability (CA) with applying adjustments to the objective function. Sample problems of size 30 to 90 are used and the results are compared with three other MIP methods. This method is shown to obtain better results for denser networks in the light of its tight LP relaxation, sparse constraint matrix, and small number of binary constraints. Bilir (2015) propose a MIP approach for large-scale TCTPs. This method is tested using a set of new samples generated using ProGen/Max random network generator. This model is able to obtain solutions for the deadline TCTP with 1000 activities and 20 time-cost alternatives. The Pareto front problem is also solved for problems with 200 activities and five time-cost modes. A more recent mixed-integer linear programming (MILP) approach is introduced by Zou, Fang, Huang, and Zhang (2016) for repetitive deadline TCTPs with fixed logic by considering multiple crews

for each activity. In addition to MILP, an alternative approximate model with less number of constraints and variables is also presented for unraveling larger problems. Test instances are generated by means of ProGen random network generator including 30, 40, 50, 80, 90, and 100 activities and CPLEX MIP Solver is used for solving these instances. MILP is shown to provide solutions for problems with up to 50 activities within the implemented computation time-frame of one hour. It is also revealed that this MILP cannot solve problems that include more than 50 activities even in three hours. Therefore, the approximate method is used for the solution of problems with up to 100 activities within the enforced maximum CPU-time of one hour. Dragović, Vulević, Todosijević, Kostadinov, and Zlatić (2017) presents an LP model for direct cost minimization of deadline TCTP. This approach is implemented using MATLAB toolbox and is applied for direct cost minimization of torrent-control projects including 10, 12, and 13 activities. The effect of daily indirect cost is not considered in this optimization procedure results of which are revealed as direct cost reductions ranging from 5.58% to 9.19%.

Dynamic programming (DP) is another category of exact procedures which aims at reducing the size of a practiced network by merging its activities. DP reduces the network size by decomposing it into a sequence of smaller sub-problems. Though, this reduction cannot be applied to a network with complex logical relationships among its activities. DP imposes a reasonable computation burden for small and simple problems; however, it might not be practical to use DP for larger and more complex networks.

Early study on DP includes Butcher's (1967) method. This approach is capable of unraveling budget TCTPs with plain series or plain parallel networks. Another DP for budget TCTP is developed by Robinson (1975). This method is also based on network decomposition. Specific assumptions and conditions are set for reducing the network into a one-dimensional problem. However, taking a multidimensional optimization approach is suggested for problems with more complex networks, due

to their difficult decomposition processes. DP method of Panagiotakopoulos (1977) implements problem simplification to solve the TCT problems by means of enumeration. De et al. (1995), giving an inclusive literature review on previous approaches, discuss the scant and sparse attention toward discrete version of TCTP. By implementing modular decomposition and incremental reduction, this thesis study introduces a centralized DP model for Pareto front TCT problems with parallel modules. Demeulemeester, Herroelen, and Elmaghraby (1996) present two DP methods for Pareto front DTCTP. The first method uses node-reduction to convert the problem network into a series-parallel network and the second model reduces the number of possible time-cost alternative combinations. Both the methods are coded in C programming language and a sample problem with 45 activities is practiced.

Other major exact methods which are classified as branch-and-bound (B&B) algorithms were first introduced by Land and Doig (1960). B&Bs are generally used for combinatorial optimization problems. B&B consists of systematic enumeration of all possible combinations of time-cost alternatives in the solution space. B&B implicitly searches only certain portions of the solution space by implementing upper and lower estimated bounds on the optimal solution. It portions the problem into subsets and is able to identify and discard those schedules that will not lead to any improvements in the objective function value.

Early B&B methods for Pareto front TCTP include horizon-varying approach of Demeulemeester et al. (1998). Lower boundaries are calculated by setting convex piecewise linear underestimations of activity time-cost curves. The quality of the underestimations is assessed using a vertical distance calculation. This method applies branching to time-cost alternatives by identifying the activity with the largest vertical distance. Branching divides time-cost options into two subset groups. This approach is implemented in Visual C++ platform and instances with 10, 20, 30, 40, and 50 activities with up to 11 time-cost alternatives are practiced.

The results are compared with Demeulemeester et al.'s (1996) method which reveal capability of this approach in providing solutions for instances with up to 30 activities and 4 time-cost options. It is also shown that nearly half of the problems including 50 activities with six or more alternatives cannot be solved using this B&B algorithm. Vanhoucke, Demeulemeester, and Herroelen (2002) present a B&B algorithm for deadline TCTP incorporating the time-switch constraints of Yang and Chen (2000) and the lower-bound calculation procedure of Demeulemeester et al. (1998). Contrary to CPM, time-switch constraints impose specific intervals for execution of the activities which are designed to deal with day, night, and weekend shifts. This algorithm is coded in Visual C++ environment and tested using a sample problem with 20 activities. Another B&B algorithm is developed by Vanhoucke (2005) for deadline TCTP with time-switch constraints. The branching process creates three child nodes as it divides the start time of activities into three sections. Branching ignores time-switch constraints of those activities that cause exceeding the project deadline. This method is implemented in Visual C++ platform and tested using instances with 10, 20, and 30 activities having up to seven time-cost alternatives. On average, this newer B&B requires less than seven seconds to unravel 30-activity sample with seven time-cost options which is four times faster than Vanhoucke et al.'s (2002) approach. A more recent B&B is generated by Degirmenci and Azizoglu (2013) for budget TCTP. This method aims to find the solution with the least cost among the solutions having the minimum completion time. This method embeds a mode elimination procedure and a network size reduction technique. Time-cost alternatives not leading to a feasible or an optimal solution are eliminated. Lower and upper-bounds for total cost are calculated using linear programming relaxation. Two procedures of LPR-based and naïve are proposed for calculation of lower-bounds. This approach is implemented in C# environment used for solution of 360 problems including up to 35 activities with up to 20 time-cost alternatives. The optimal solutions are found by means of CPLEX 10.1.

Benders decomposition (BD) is another exact procedure which was first introduced by Benders (1962). It is generally used for solution of large-scale stochastic/linear programming problems which partitions the practiced problem into multiple simpler sub-problems. Benders decomposition consists of two stages, in first of which a lower-bound is set and the master problem is solved for a subset of variables and in the second stage, an upper-bound is set and the remaining variables are calculated for a sub-problem using the values determined in the preceding stage. If an infeasibility of first stage decision is identified in the latter stage, additional constraints are used to solve the master problem.

Hazır, Haouari, and Erel (2010) propose a BD method for budget TCTP by modifying the original method of Benders. It includes an improved decomposition approach and as well as a branch-and-cut procedure. Problems networks with 85 to 136 activities are solve using this procedure. It is shown that this BD can outperform both Benders and IBM's CPLEX 9.1 methods. Another BD was presented by Hazır, Erel, and Gunalay (2011) for deadline TCTP with uncertain cost parameters. Three alternative optimization models are proposed that assume interval uncertainty for unknown parameters. This model assumes certain start and finish times for activities as their time-cost alternatives are associated with fixed durations. All the uncertainty is reflected on the cost of an activity using probabilistic intervals. This method is tested using problems with 85 to 136 activities.

2.4. Heuristic methods for TCTP

In contrast to the exact procedures, heuristic methods are convenient tools of optimization, requiring non-substantial computation time and resources. Some of the heuristics can be implemented even short of a computer's assist. Derived from the Greek word "Heuriskein" meaning "to find", heuristics involve simple rules for finding solutions to difficult optimization problems. Nonetheless, the optimality is not guaranteed in the heuristic methods. The solutions obtained by means of

heuristics are rather satisfactory since they are either optimal or near-optimal. The constructive and the improvement heuristics are the most revered variants of the heuristic algorithms. The former uses a stepwise procedure to generate solutions, generating them one at a time until a feasible solution is met. Generally, a feasible solution is not obtained in the course of the construction heuristics unless the conclusion of the procedure is reached. The latter type of the heuristic algorithms i.e. the improvement heuristics, initiate with a feasible solution and successively improve it via a series of modifications. In the course of this procedure, usually a feasible solution is preserved regardless of the progression of the process (Aminbakhsh, 2013). Existing studies providing heuristic algorithms acknowledge that they are problem dependent and cannot handle large-scale problems efficiently (Siemens, 1971). Most of the present heuristics assume merely linear time-cost functions and they fail to solve the Pareto front problem (Feng et al., 1997; Zheng et al., 2005).

Early heuristics in the literature of TCTP include the procedure proposed by Siemens (1971). Known as SAM (Siemens Approximation Method), a logical systematic procedure is developed to minimize the overall project cost which is suited for both manual and computer aided calculations. It is capable of obtaining solution for convex nonlinear TCTPs by making multiple piecewise linear curve approximations. The procedure initiates with the construction of the project network, thereby, crashing critical activities one at a time based on some specific rules. Of these rules, the most important is selection of a critical activity with minimum amount of cost-slope. The act of crashing continues until either all the activities are crashed, or those with normal duration have cost-slopes greater than the daily indirect cost. The major drawback related to this approach is that it requires determination of all the critical paths in the network which might prove expensive to solve complex problems. A sample problem with eight activities is fitted into the model and it is shown that the results compare well with the results of a linear programming approach. Goyal (1975) proposes a modified version of

SAM which uses effective cost-slopes instead of the plain cost-slopes and is capable of obtaining solutions for TCTPs with convex nonlinear cost-slopes. This method also introduces a de-shortening technique which is applied to the excessively shortened paths. The same eight activity network of Siemens (1971) is used to test the proposed heuristic. An alternative heuristic is proposed by Moselhi (1993) to minimize the overall project cost which involves replacing CPM network with a corresponding structure. Using the analogy between the support settlement of structural analysis and the completion deadline of TCTPs, the overall cost of schedule compression is calculated using the sum of all member forces. Aminbakhsh (2013) and Aminbakhsh and Sonmez (2016, and 2017) develop a new cost-slope-based heuristic approach for their hybrid optimization method which is regarded as one of the first TCTP heuristics operating in discrete search space. Named as modified-SAM, this method is designed for problems with discrete time-cost relationships. Modified-SAM, unmatched by any other previous heuristic for TCTP, is capable of obtaining multiple non-dominated solutions in a single run for Pareto front problem. Bettemir and Birgonul (2017) explain another discrete heuristic method called Network Analysis Algorithm (NAA) for deadline and cost minimization TCTP. Least cost-slope activities are crashed under three conditions of single critical path, multi-critical path, and necessity of crashing non-critical activities. This approach uses an elimination algorithm to reduce the number of crashing options by excluding those critical activities that are shared with all the critical paths. 18-activity problem of Feng et al. (1997) and 63-activity problem of Sonmez and Bettemir (2012) are used for performance measurement of NAA. Compared with hybrid-GA of Sonmez and Bettemir (2012), NAA is concluded to require one percent of meta-heuristics computation burden and contributing to higher accuracy. Su, Qi, and Wei (2017) propose an equivalent simplification-based approach for solution of nonlinear continuous deadline TCTP. This algorithm transforms TCTP into a simple activity float problem then uses a polynomial algorithm to unravel the transformed problem. Both the simplification and polynomial algorithms are implemented using LINDO optimization software. This

method narrows the domain of direct cost for each activity duration to achieve higher efficiency and accuracy. Redundant activities are also discarded to reduce the scale of the TCTP. This algorithm is experimented on a 116-activity problem which requires 30 minutes of processing time in the absence of the simplification method, and takes 2 seconds when the problem is simplified to a 33-activity network.

2.5. Meta-heuristic methods for TCTP

Inspired by the stochastic occurrences of the nature, evolutionary algorithms are among the meta-heuristics practiced in solution of DTCT problems. “Meta”, meaning “beyond” is an indication of higher-level algorithms when compared with the heuristics. This is because of the fact that heuristics are problem dependent while meta-heuristics are independent of the nature of the problem. Existing meta-heuristics unravel an optimization problem by randomly searching the solution space. Contrary to heuristic methods, meta-heuristics are designed to explore a problem in an iterative fashion. This is mainly performed in view of prevention of meta-heuristic from getting stuck into the local optima. Ironically, the main deficiency of the existing meta-heuristics is observed as their inability to escape from local optima (Zheng et al., 2005; Sonmez and Bettemir, 2012). In addition, similar to the heuristics, meta-heuristics are incapable of securing the optimality of the solutions; rather, they can provide near-optimal solutions within only moderate computational effort. This category of optimization methods is well associated with the modern studies, including genetic algorithms (GA), ant colony optimization (ACO), particle swarm optimization (PSO), shuffled frog leaping (SFL), and simulated annealing (SA).

Of the meta-heuristic methods, Genetic algorithms (GA) are the most popular and highly practiced approaches. GA was first introduced by Holland (1975) which as the name implies, is inspired by the natural selection and genetic reproduction

process. Feng et al. (1997) present a GA for solution of Pareto front TCT problem. Addressing the lack of sound methods for coping with large-scale TCTPs, a GA is generated in which normal and crash modes of an activity are defined as two chromosomes. This method develops new solutions by iterative cross-overs and mutations, fitness values of which are evaluated using their minimal distances to a convex hull. Genetic drift (raised by Goldberg and Segrest, 1987) is avoided by retaining each string for the next generation. An 18–activity problem is used to test the proposed model, different variants of which has since been used extensively within the TCTP literature. It is shown that this method is capable of locating more than 95% of the non-dominated solutions for the benchmark problem. Li and Love (1997) introduce a GA based approach for deadline TCTP with improved mutation and crossover processes. A sample problem of 10 activities is fitted into both the proposed and the original GAs and the results are compared. It is indicated that the presented method outperforms the original GA due to its higher search efficiency.

Hegazy (1999) comes up with another GA for deadline TCTP. This method requires shorter computation time as it employs the same techniques proposed by Li and Love (1997). This GA is implemented as a macro in Microsoft Project 1995 and is tested using instances with 18, 36, 108, and 360 activities. It is concluded that although this method is capable of obtaining solutions for all the practiced instances, the computation time increases as the problem becomes more complicated. Meanwhile, the core 18-activity problem is actually the same instance introduced by Feng et al. (1997) with a slight difference in one of its activities. Zheng et al. (2005) introduce a GA model for Pareto front TCTP. A modified adaptive weight approach (MAWA) is used to adjust the priority of each objectives of time and cost with regard to performance of preceding generation. As the generations evolve, MAWA administers a decreasing pattern for the mutation rate to prevent premature convergence. Pareto ranking and niche formation are also implemented to this model with the former serving as a selection criterion and the latter as a population diversifier. This approach is implemented as a macro in

Microsoft Project 2000 and is practiced using Feng et al.'s (1997) 18-activity problem. The results indicate robustness of this method and it is shown that the results keep improving as generation count is increased beyond 300.

Kandil and El-Rayes (2006) present a GA with two parallel processing techniques. Global parallel and coarse-grained GAs are used for solution of Pareto front TCTPs. Instances with 180, 360, and 720 activities are generated by using the time-cost alternatives of Feng et al. (1997) and by copying a slightly modified version of the 18-activity network of Feng et al. (1997) in serial several times. Global parallel GA is able to find 267 solutions for the 180-activity problem in 42.6 minutes. The coarse-grained GA, on the other hand, achieves 68 Pareto solutions in 11.4 minutes. Similarly, global parallel GA is able to find 232 solutions for the 360-activity problem in 173.4 minutes. The coarse-grained GA, on the other hand, achieves 94 Pareto solutions in 30.6 minutes. 136.5 hours on a single processor, and 15.4 hours over a supercomputing cluster of 50 processors is required for global parallel GA to achieve 303 Pareto front solutions for the 720-activity problem and the coarse-grained GA achieves 132 non-dominated solutions in 118.18 minutes. Eshtehardian et al. (2008) discuss a GA which can reflect the uncertainties in time and cost of project activities using the fuzzy sets theory of Zadeh (1965). This method can obtain solution for stochastic Pareto front TCTP by assigning triangular fuzzy numbers to time and cost of the work-packages. Hamming-distance and Euclidian-distance techniques are implemented to GA and α cut approach is used in ranking the non-dominated solutions in accord with the decision maker's acceptance of risk level. The GA based approach is tested using the 18-activity problem of Feng et al. (1997).

Eshtehardian, Afshar, and Abbasnia (2009) demonstrate another fuzzy-based multi-objective GA for Pareto front optimization of TCTPs with uncertainties in their activity time-cost pairs. This model addresses risk acceptance level and degree of optimism of a decision maker employing α cut approach and optimism index,

respectively. Fuzzy set theory is explicitly embedded into this optimization process which ranks fuzzy numbers using an approximate method in reference to the left and right dominance. This GA model uses the general concepts of NSGA-II combined with single cut cross over, uniform mutation, and tournament selection procedures. Time-cost options of 7-activity problem of Zheng et al. (2005) and 18-activity problem of Feng et al. (1997) are defined as triangular fuzzy numbers. These instances are fed into the GA, deterministic results of which are compared with the original studies mentioned. Sonmez and Bettemir (2012) present a hybrid-GA for both the deadline and cost minimization TCTPs. This method combines the complementary capabilities of simulated annealing (SA) and quantum simulated annealing (QSA) together with the GA approach. This method is coded in Visual C++ environment, SA module of which aims to improve hill-climbing, while its QSA module aims to improve local search capabilities of the hybrid model. Problem sizes ranging from 18 to 630 activities are fitted into the model and the results are compared with the optimal solutions achieved by mixed integer programming technique implemented in AIMMS optimization software. Test instances are based on the 18-activity problem of Feng et al. (1997), 29-activity problem of Chassiakos and Sakellaropoulos (2005), and a hypothetical 63-activity problem. Outputs of paired t-test is given which verifies improved convergence capability of this model against a typical GA.

Mungle, Benyoucef, Son, and Tiwari (2013) demonstrate a fuzzy clustering-based GA for Pareto TCTPs with uncertainties in their time-cost amounts. The anticipated qualities of execution alternatives are measured by means of analytical hierarchy process (AHP). This approach combines NSGA-II with external repository concept and linkage-based hierarchical clustering technique for storing and managing diversity of the Pareto solutions, respectively. Implemented in MATLAB, this GA incorporates crowded-comparison, crowding distance, two-point crossover, and bit-flip mutation operators. This model is used for analyzing 7-activity problem of Burns, Liu, and Feng (1996), a hypothetical 12-activity problem, and 18-activity

problem of Feng et al. (1997). Findings of this approach is compared with HS of Geem (2010), SPEA-II of Zitzler, Laumanns, and Thiele (2001), and exact method of Burns et al. (1996). Generational distance (GD) and Range Variance (RV) metrics are used for measuring convergence and diversity of the obtained Pareto solutions, respectively. These metrics are then measured for PSO of Zhang and Li (2010) and GA of Feng et al. (1997) to carry out comparisons. Zhang, Zou, and Qi (2015) introduce a GA for repetitive projects with soft logic. A mixed-integer nonlinear programming approach is used to combine TCTP and the soft logic concept which allows for changes to be made in the logical relationships, in view of a faster and cheaper project realization. This approach is tested using a sample problem with five activities including a single resource for execution of each of its activities.

Koo, Hong, and Kim (2015) demonstrate GA-based integrated multi-objective optimization (iMOO) model for Pareto front TCTP. This model is able to carry out optimization using four different fitness functions including trade-offs between time-cost, cost-quality, sustainability-cost, and productivity-safety. Prior to GA phase, heuristic module of this model defines the minimum and maximum extreme points in the solution space. OptQuest tool of Crystal Ball is used to implement the GA procedure. Capabilities of this approach for the time-cost objective function is tested using the 7-activity problem of Zheng et al. (2005). It is declared that this model is able to find the Pareto front identical to the previous studies. Huang, Zou, and Zhang (2016) develop another GA-based model for repetitive deadline TCTPs with soft logic by considering a single crew for each activity. They aim to reduce the overall cost of project by implementing a proper sequencing. They use linear programming for reducing a repetitive construction projects to a classical TCTP, afterwards, they use a GA to tackle the reduced problem which is programmed in MATLAB. This approach is tested using a 5-activity problem solved considering two sequencings as well as a 6-activity project solved considering three sequencing strategies. El-Abbasy, Elazouni, and Zayed (2016) develop an elitist non-dominated

sorting GA for Pareto front TCTP that considers multiple projects with multiple objectives. This method unravels TCTP by taking into account the financing cost, maximum required credit, profit, and resource levelling of the project. Of the three models implemented, scheduling model is validated using the 18-activity problem of Feng et al. (1997). This model which is also able to tackle pure TCTPs is shown to locate 31 non-dominated solutions for this instance which are compared visually to the results of Feng et al. (1997).

A more recent and notable GA-based approach is introduced by Agdas, Warne, Osio-Norgaard, and Masters (2018) for large-scale cost minimization and deadline TCTPs. Parallel GA is developed by applying the Scalable Concurrent Operations method to the open source Distributed Evolutionary Algorithms framework of Python programming language. A custom GA is developed employing a new problem encoding, implementing weighted graph method instead of iterative CPM calculations, a parallel fitness evaluation technique, and a new stagnation criterion. This model is shown to run four times faster in the light of parallel computing on a high-performance computing facility with eight CPU cores. Combined with their modifications, this GA is proved to be 100 times faster than a typical GA running on the same computer on a single CPU core. Large-scale problems with 630, 1800, 3150, and 6300 activities are practiced and the results are compared with Sonmez and Bettemeir (2012). The 1800-activity problem is based on the 18-activity network of Feng et al. (1997) and the remaining instances are based on the 63-activity problem of Sonmez and Bettemir (2012) which are generated by copying the same network in serial several times. Performance of the parallel GA is evaluated with respect to its accuracy and processing time in two stages; viz., before and after applying the modifications. It is contended that the results of unmodified parallel GA exceed the accuracy of Sonmez and Bettemir's (2012) hybrid-GA approach for the 630-activity problem; though, it requires more than four hours to unravel this project. The accuracy of the modified version however is demonstrated to be on par with hybrid-GA of Sonmez and Bettemir (2012) with much less runtime

requirements. CPU time for the larger problems is reported to vary from 5.82 to 16.76 hours with average deviations ranging from 4.73% to 14.72%.

An alternative meta-heuristic approach covers Ant Colony Optimization (ACO) methods. ACO was first proposed by Colorni, Dorigo, and Meniezzo (1992) which is inspired by the coordinated interactions of ant colonies in search for sources of food. Ng and Zhang (2008) present an ACO based model for Pareto front TCTP. This model is equipped with modified adaptive weight approach of Zheng et al. (2004) for fitness function evaluations. This model is programmed in Visual Basic platform and its results on the 18-activity problem of Feng et al. (1997) are compared with Elbeltagi, Hegazy, and Grierson's (2005) typical ACO method. This method is shown to decrease the computation resource requirements of the original ACO. Xiong and Kuang (2008) develop another ACO employing the modified adaptive weight approach of Zheng et al. (2004) for Pareto front TCTP. This method incorporates two selection criteria for the alternatives. The first selection is made regarding a maximization criterion and the second involves a probability distribution function. The performance of this method is evaluated using 7-activity problem of Zheng et al. (2004) and the 18-activity of Feng et al. (1997). This method is capable of obtaining solutions by exploring rather smaller portion of the solution space.

Afshar et al. (2009) develop a multi-colony non-dominated archiving ACO for Pareto front TCTP. Separate ant colonies are assigned to time and cost objectives. These separate colonies are designed to share solutions to be evaluated with regard to the counteracting objective. Performance of this method is measured using the 18-activity problem of Feng et al. (1997) under three assumptions regarding the amount of daily indirect cost. For all the cases, the results prove to be better than the solutions achieved by ACO-based and GA-based methods of Zheng et al. (2005). Kalhor, Khanzadi, Eshtehardian, and Afshar (2011) explain a fuzzy-based non-dominated archiving ACO for Pareto front optimization of TCTPs with

uncertainties in their activity time-cost pairs. In this model α cut approach accounts for the risk acceptance level of the decision maker. Degree of optimism of a decision maker is also embedded to this method using an optimism index. Left and right dominance rule is employed for conducting comparison in the non-dominated sorting process. Coded in MATLAB, this ACO uses one colony for minimization of fuzzy cost and another for minimization of fuzzy duration. The iteration best solution inside the colonies and the best-so-far solution are used for pheromone trail updates. 18-activity network of Feng et al. (1997) with fuzzy time-cost pairs of Eshtehardian et al. (2009) is practiced by assuming non-symmetric triangular shape for each activity execution alternative. The extent of convergence and spread of the obtained Pareto front is assessed using Euclidean distance and delta (Δ) metrics, respectively. Findings of this model are demonstrated to be fully compatible with outcomes of Eshtehardian et al. (2009). Zhang and Ng (2012) propose another ACO approach for cost minimization and Pareto front TCTPs. This method is implemented in VBA and is integrated into Microsoft Project. 18-activity problem of Feng et al. (1997) is practiced and the results are compared with GA-based method of Zheng et al. (2005).

One other domain of meta-heuristics includes the Particle Swarm Optimization (PSO) algorithms. Developed by Kennedy and Eberhart (1995), PSO is a population-based algorithm imitating the choreography of bird flocks that communicate together as they fly. Elbeltagi et al. (2005) studies performance of five different evolutionary algorithms for cost minimization TCTP, one of which is a PSO-based model. The five meta-heuristics include GA, memetic algorithm (MA), PSO, ACO, and shuffle frog leaping (SFL) all of which are coded in Visual Basic platform. The practiced test instances also contain the 18-activity network of Feng et al. (1997). The results of this comparison reveal PSO outperforming other techniques based on the solution quality and placing second regarding the computation time requirements. Yang (2007b) presents a modified PSO for Pareto front TCTPs. Pbest is updated in case the solution is strongly dominating and gbest

accepts only the particle that dominates fewest solutions. Non-dominated solutions are stored and updated in a separate archive of elite solutions. Indirect cost is not included in the optimization process; however, it is added to the final Pareto front solutions. Performance of this algorithm is measured for solution of a 14-activity problem. Yang (2007a) proposes another PSO method for Pareto front TCTP. This model is coded in MATLAB programming environment and the indirect cost is implemented exogenously to the final Pareto front solutions. A hypothetical example with 8 activities and a case problem with 28 activities is practiced to test the algorithm. Small average percent deviation (APD) amounts from the optima is reported for this approach.

Zhang and Li (2010) introduce a PSO for Pareto front TCTP. Sparse-degree and roulette-wheel selection are implemented to improve diversity of particles and convergence capabilities of the model. Gbest is determined using the combined scheme used in this model which is coded in Visual C++ environment. Results of this model for a 7-activity and an 18-activity (of Feng et al., 1997) problem is shown to best GA of Feng et al. (1997). Fallah-Mehdipour, Bozorg Haddad, Rezapour Tabari, and Mariño (2012) explain revised multi-objective PSO and GA models for Pareto front TCTP. Both methods are modified for problems with discrete domains, e.g., PSO is provided with a classification technique which divides a feasible continuous space into discrete units and the continuous decision is rounded to the closest integer value. Capabilities of these two approached are assessed and compared using the 18-activity problem of Feng et al. (1997). PSO and GA are compared with each other with respect to Generational Distance (GD) of their results also using each method's share of non-dominated solutions in the final Pareto front; GA is revealed to be more successful for solution of the practiced instance. Aminbakhsh and Sonmez (2016) propose a discrete PSO for cost minimization and deadline TCTP. This method is based on the novel principles for representation, initialization and position-updating of the particles which facilitate adequate representation of the discrete search space and enhance accuracy due to

improved quality of the initial swarm. This method uses semi-deterministic initialization scheme by employing the modified-SAM (Aminbakhsh, 2013) heuristic. This approach is implemented in C++ and tested using benchmark problems and large-scale instances generated using ProGen/Max network generator. Different variants of 18-activity problem of Feng et al. (1997), 63-activity and 630-activity problems of Sonmez and Bettemir (2012), and large-scale problems including 200 and 500 activities with two to five time-cost options are used for performance measurement of this procedure. Findings of this approach is compared with outcomes of numerous previous studies and the deviations from the optimal solutions are calculated using exact solutions of a MIP method.

In a more recent study, Aminbakhsh and Sonmez (2017) present a Pareto front PSO for Pareto oriented optimization of large-scale TCTPs. This model is treated with unique particle representation, initialization, and position-updating techniques. This multi-objective PSO is also embedded with a modified version of SAM which is named simplified-SAM; though, unlike any other variant of SAM, instead of determining the least-cost-slope critical activities for all of the multiple parallel critical paths, one critical activity at a time is crashed to circumvent the bottleneck of finding all the critical activities of the network. This PSO incorporates the external repository concept and redefines position update and *gbest* calculation routines. C++ is used for coding of this approach. This model is experimented on different variants of 18-activity problem of Feng et al. (1997), 180-activity, 360-activity, and 720-activity problems of Kandil and El-Rayes (2006). Pareto fronts obtained by this approach are compared with the results of existing methods and the deviations from the true Pareto fronts are calculated for some of the practiced instances by means of a MIP method.

Another meta-heuristic approach is categorized as Shuffle Frog Leaping (SFL) algorithm. SFL imitates the behavior of a group of frogs locating a source with the maximum amount of available food which was originally developed by Eusuff and

Lansey (2003). Elbeltagi, Hegazy, and Grierson (2007) explain an SFL for cost minimization TCTP. The original SFL is modified by implementing a search-acceleration parameter to avoid getting stuck into the local optima. This parameter is designed to sustain the balance between the local and the global search of the algorithm. The proposed SFL is implemented as a VBA code in Microsoft Project. The performance of this model is validated using benchmark problems as well as instances with 18, 90, and 180 activities all of which are based on the 18-activity network of Feng et al. (1997). Results reveal less computation time requirement and high success rate of this model compared with the original SFL and a GA algorithm.

Of the meta-heuristic algorithms, Simulated Annealing (SA), being first proposed by Kirkpatrick, Gellat, and Vecchi (1983), is inspired by the heating and controlled cooling of alloys subject to tempering. At higher temperatures, the molecules move freely in any direction which can be restricted by lowering the temperature. Anagnostopoulos and Kotsikas (2010) analyze five variants of SA for cost minimization of TCTP. Visual Basic is used to implement SA and problems with 100, 200, and 300 activities are practiced for performance evaluation. All the instances are generated by means of RanGen random network generator. Analysis of variance (ANOVA) and Duncan Multiple Range Test are used to measure the performance and to rank the five SA algorithms.

An alternative meta-heuristic algorithm is developed by Vanhoucke and Debels (2007). TCTP is exerted considering time/switch constraints, work continuity constraints, and net present value maximization. This method is coded in Visual C++ platform and is a hybrid of a heuristic and a truncated dynamic programming technique. The first module undertakes neighborhood search and diversification steps and the second module relaxes non-critical activities. The quality of the solutions obtained by this method is measured using the B&B algorithm of Demeulemeester et al. (1998).

Geem (2010) introduce Harmony Search (HS) meta-heuristic algorithm for Pareto front TCTP which is based upon the analogy between musician's experience and the optimization process. This method involves a phenomenon-mimicking approach which is programmed in Microsoft Excel. Unlike GA that uses two solutions vectors, this method considers all the solutions in generating a new one. Performance of this model is measured using the 7-activity problem of Zheng et al. (2005) and the 18-activity network of Feng et al. (1997). The provided results stack up well against GA of Zheng et al. (2004) and ACO of Xiong and Kuang (2008).

Abdel-Raheem and Khalafallah (2011) come up with another meta-heuristic method hailed as Electimize for cost minimization TCTP. This approach simulates the behavior of electrons as they tend to move through a multi-branch electric circuit via the branch with the least resistance. Each solution is represented by a wire, segments of which represents the activity options. Coded as a macro in Microsoft Project, Electimize uses Ohm's law and Kirchhoff's rule through its simulation process which assess quality of each solution using their global resistance value. This method is applied for solution of the 18-activity problem of Feng et al. (1997).

In a similar fashion, Vanhoucke (2015) develops Electromagnetic (EM) model for four different variants of deadline TCTP. Time/switch constraints, work continuity constraints, and net present value maximization are also considered in optimization of the deadline problem. EM relies on the law of Coulomb and uses the principles of Birbil and Fang (2003) to calculate charges and forces. This method incorporates two local searches; one for crashing and one for relaxing activities. Projects with 10, 20, 30, 40, and 50 activities with up to 11 time-cost options are fitted into the model which is coded in Visual C++ platform. Results are compared with the exact solutions obtained using the procedures described in Demeulemeester et al. (1998) and Vanhoucke (2005) and by using LINDO optimization software. Results of this method is contended to best Vanhoucke and Debels's (2007) solutions.

The major existing exact, heuristic, and meta-heuristic approaches for TCTP are summarized in Table 2.1. Entries are arranged in a chronological order in this table, including the practiced problem types, implemented platforms, size of the instances used with their daily indirect cost rate (IC), computation time of the proposed methods (CPU time), their number of consecutive runs, and their associated average percent deviation from optima (APD). The last column of this table gives a brief explanation and highlights remarkable points of each study. Unreported details are tabulated as 'na' in Table 2.1. Different TCT problem types are abridged as follows in this table:

- TCT: Cost minimization problem;
- TCT-BT: Budget problem;
- TCT-DL: Deadline problem;
- TCT-PF: Pareto front problem;
- St. TCT-DL: Stochastic deadline problem;
- St. TCT-PF: Stochastic Pareto front problem.

Table 2.1 – Existing exact, heuristic, and meta-heuristic methods for solution of TCTP.

Year	Author(s)	Method	Problem	Platform	# of Acts.	IC	CPU Time (s)	# of Runs	APD (%)	Remarks
1967	Crowston and Thopson	MIP	TCT-DL	na	8	na	5	na	na	A mathematical model for Decision CPM is presented to carry out path reductions.
1971	Siemens	Heuristic	TCT	na	8	na	na	na	na	Logical systematic approach involving a number of rules for expediting the activities that incur least additional costs. Since this method only considers minimum cost-slope for crashing activities, it might shorten the duration beyond required amount.
1975	Robinson	DP	TCT-BT	na	na	na	na	na	na	Based on network decomposition, this method sets specific assumptions and conditions for reducing the network into a one-dimensional problem.
1995	De, Dunne, Ghosh, and Wells	DP	TCT-PF	na	5 10	na	na	na	na	A centralized approach for deadline problem with no parallel modules and a combination of modular decomposition with incremental reduction approaches for problems with parallel modules. This method is only effective for networks with reasonably low values of certain parameters.
1995	Liu, Burns, and Feng	Hybrid LP/IP	TCT-DL	VBA Macro	7	na	1800	na	na	The hybrid approach uses LP to set lower-bounds for cost and uses IP to find the exact solutions.
1996	Demeulemeester, Herroelen, and Elmaghraby	DP	TCT-PF	C	45	na	530.4	na	na	Two DPs are presented first of which uses node-reduction to convert the network into a series-parallel and the second method reduces the number of possible time-cost combinations.

Table 2.1 – Existing exact, heuristic, and meta-heuristic methods for solution of TCTP (*Continued*).

Year	Author(s)	Method	Problem	Platform	# of Acts.	IC	CPU Time (s)	# of Runs	APD (%)	Remarks
1997	Feng, Liu, and Burns	GA	TCT-PF	na	18	$0 $1500	na	na	na	A GA that calculates fitness values exploiting minimal distance to convex hull and retains each string for next generation to avoid genetic drift. This approach only tackles Finish-to-Start relationships neglecting possible resource constraints.
1997	Li and Love	GA	TCT-DL	na	10	na	na	na	na	A modified GA algorithm with improved mutation and crossover processes.
1998	Demeulemeester, Reyck, Foubert, Herroelen, and Vanhoucke	B&B	TCT-PF	Visual C++	10 20 30 40 50	na	0.34 19.17 58 105.4 127.56	na	na	Horizon-varying approach is embedded into branch and bound method and qualities of lower boundary underestimations are assessed by vertical distance computations. Effectiveness and efficiency of this model decreases significantly for larger networks with multiple modes.
1999	Hegazy	GA	TCT-DL	VBA Macro	18	$200	390	na	8.139	A premier GA model that minimizes total cost while taking into account the project-specific constraints on time. Random nature of model requires long processing time in VBA environment.
2002	Vanhoucke, Demeulemeester, and Herroelen	B&B	TCT-DL	Visual C++	20	na	na	na	na	A B&B with time-switch constraints is presented which is designed to deal with day, night, and weekend shifts.

Table 2.1 – Existing exact, heuristic, and meta-heuristic methods for solution of TCTP (*Continued*).

Year	Author(s)	Method	Problem	Platform	# of Acts.	IC	CPU Time (s)	# of Runs	APD (%)	Remarks
2004	Moussourakis and Haksever	MIP	TCT-DL	na	7	na	na	na	na	A method able to solve deadline and budget TCT problems with linear, piecewise linear, or discrete objective functions. Requires no network notation system and makes minimal assumptions regarding the type of TCT functions. Since this method requires substantial computational resources, it suits small to medium instances.
2005	Chassiakos and Sakellaropoulos	MIP	TCT-PF	LINDO	29	€1200	<1	na	na	Exact and approximate models are developed and compared for solution of a hypothetical problem.
2005	Elbeltagi, Hegazy, and Grierson	GA	TCT	Visual Basic	18	$500	16	20	2.171	Five meta-heuristic algorithms (GA, MA, PSO, ACO, and SFL) are compared, revealing poor performance of GA as well as robustness of PSO methods.
		MA					21		0.759	
		PSO					15		0.415	
		ACO					10		3.351	
		SFL					15		2.96	
2005	Vanhoucke	B&B	TCT-DL	Visual C++	10	na	0.003	na	0	Branching process of this B&B algorithm ignores time-switch constraints of those activities that cause exceeding the project deadline. Lower-bound branching is applied to activities having greater than duration lead times.
					20		0.010 to 0.089		0	
					30		0.089 to 6.535		0.015	
2005	Zheng, Ng, and Kumaraswamy	MAWA-GA with Niche Formation	TCT-PF	na	18	$1500	na	na	1.324 (Based on 4 Obs.)	A GA that recruits MAWA, Pareto ranking, and Niche formation to avoid genetic drift, administer selection pattern, and exert diversifier, respectively.

Table 2.1 – Existing exact, heuristic, and meta-heuristic methods for solution of TCTP (*Continued*).

Year	Author(s)	Method	Problem	Platform	# of Acts.	IC	CPU Time (s)	# of Runs	APD (%)	Remarks
2006	Kandil and El-Rayes	GP-GA	TCT-PF	na	180	$0	2,556 to 14,688	na	na	A GA with two parallel processing techniques of global parallel and coarse grained is presented and used for Pareto oriented optimization of problems with up to 720 activities. Despite parallel computing, there remains the computation time problem since both variants require significant processing times.
					360		10,404 to 75,096			
					720		55,296 to 491,400			
		CG-GA			180		684			
					360		1,836			
					720		7,092			
2007	Elbeltagi, Hegazy, and Grierson	MSFL	TCT	VBA Macro	18	$500	8	20	0	An SFL algorithm that incorporates a time-variant parameter to avoid falling into local optima.
					90		810	na	na	
					180		na			
2007	Vanhoucke and Debels	Meta-heuristic	TCT	Visual C++	10	na	0.008	na	0.037	Heuristic portion involves neighborhood search and diversification steps. The second portion of algorithm uses truncated dynamic programming to relax non-critical activities.
					20		0.096		0.05	
					30		0.337		0.044	
					40		0.811		0.098	
					50		1.605		0.114	
2007a	Yang	PSO	TCT-PF	MATLAB	8	$500	48	na	0.406	A PSO algorithm capable of handling any function type which requires manual calculations for subsequent "what if" analysis. Indirect costs are not provisioned throughout the optimization process.
					28	na	600		na	

Table 2.1 – Existing exact, heuristic, and meta-heuristic methods for solution of TCTP (*Continued*).

Year	Author(s)	Method	Problem	Platform	# of Acts.	IC	CPU Time (s)	# of Runs	APD (%)	Remarks
2007b	Yang	PSO	TCT-PF	na	14	$600	na	na	na	Selection process of PSO method is modified in favor of strongly dominant solutions as pbest, and solutions of scant areas as gbest. Indirect costs are not included explicitly in the optimization process.
2008	Eshtehardian, Afshar, and Abbasnia	Fuzzy-GA	St. TCT-PF	na	18	$0	na	na	0 (Based on 4 Obs.)	Fuzzy set theory enables GA to handle stochastic TCTPs.
2008	Ng and Zhang	ACS-TCO	TCT-PF	Visual Basic	18	$500 $1500	na	na	0 0.018	MAWA is integrated into ACO. This model is susceptible to premature convergence with higher iterations and it is too sensitive to selection of parameters.
2008	Xiong and Kuang	MAWA-ACO	TCT-PF	na	7 18	na $1500	na	na	na	MAWA is embedded into ACO and selection of the options are made according to membership of a random variable, the first selection involving a maximization criterion, and the other incorporating a probability distribution function.

Table 2.1 – Existing exact, heuristic, and meta-heuristic methods for solution of TCTP (*Continued*).

Year	Author(s)	Method	Problem	Platform	# of Acts.	IC	CPU Time (s)	# of Runs	APD (%)	Remarks
2009	Afshar, Ziaraty, Kaveh, and Sharifi	NA-ACO	TCT-PF	na	18	$0	na	na	0 (Based on 44 Obs.)	Multi-colony non-dominated archiving ACO that assigns separate ant colonies to each objective and evaluates the found solutions respecting the competing objective within the next colony.
						$200			0 (Based on 18 Obs.)	
						$1500			0 (Based on 4 Obs.)	
2009	Eshtehardian, Afshar, and Abbasnia	Fuzzy-MOGA	St. TCT-PF	na	7	$500	na	na	na	Fuzzy numbers comparison and GA are utilized to locate PF based on the project manager's risk acceptance level and degree of optimism. This model is operational only for project managers familiar with the concepts of uncertainty and risk management.
					18	$200	900		0 (Based on 20 Obs.)	
2010	Anagnostopoulos and Kotsikas	SA	TCT	Visual Basic	100	1%, 2%, and 3% of Total Cost	0.18 to 0.45	30	na	Performance of five variants of SA algorithm are analyzed and compared with each other.
					200		0.99 to 1.92			
					300		1.20 to 3.29			

Table 2.1 – Existing exact, heuristic, and meta-heuristic methods for solution of TCTP (*Continued*).

Year	Author(s)	Method	Problem	Platform	# of Acts.	IC	CPU Time (s)	# of Runs	APD (%)	Remarks
2010	Geem	HS	TCT-PF	MS Excel	7 18	$1500 $0	na 5	na	0 (Based on 5 Obs.) 2.27 (Based on 19 Obs.)	Compared with GA, all solution vectors are considered in generating a new vector rather than two; it does not require binary-to-decimal conversion.
2010	Hazir, Haourai, and Erel	BD	TCT-BT	na	136	na	na	na	na	Original BD is modified with an improved decomposition approach and as well as a branch-and-cut procedure.
2010	Zhang and Li	CSMO-PSO	TCT-PF	Visual C++	7 18	$0	63.66 205.08	50	na	A multi-objective PSO that combines sparse-degree and roulette-wheel selection for determining the gbest of each particle.
2011	Abdel-Raheem and Khalafallah	Electimize	TCT	VBA Macro	18	$500	na	20	0	This method simulates the behavior of electrons moving through electric circuit branches with the least resistance.
2011	Hazir, Erel, and Gunalay	BD	St. TCT-DL	na	136	na	19139.61	na	na	Three optimization models that assume interval uncertainty for unknown parameters are presented. All the uncertainty is reflected on the cost of an activity using probabilistic intervals.
2011	Kalhor, Khanzadi, Eshtehardian, and Afshar	NA-ACO	St. TCT-PF	MATLAB	18	$200	29.42	10	0 (Based on 18 Obs.)	α-cut approach accounts for accepted risk level and β concept is used for ranking solutions; left and right dominance ranking method is used for non-dominated sorting.

Table 2.1 – Existing exact, heuristic, and meta-heuristic methods for solution of TCTP (*Continued*).

Year	Author(s)	Method	Problem	Platform	# of Acts.	IC	CPU Time (s)	# of Runs	APD (%)	Remarks
2012	Fallah-Mehdipour, Bozorg Haddad, Rezapour Tabari, and Mariño	MOPSO	TCT-PF	na	18	$0	na	na	0.944 (Based on 1 Obs.)	Discrete versions of MOPSO and NSGA-II are compared for solution of Pareto front TCTP. NSGA-II is concluded to be more successful.
2012	Sonmez and Bettemir	HA	TCT-DL	Visual C++	18	$200	na	10	0	A hybrid GA combining potencies of SA along with quantum simulated annealing (QSA).
					29	€1200			0	
					290	€1200			0.43	
			TCT		18	$1500			0	
					29	€1200			0	
					63	$2300			2.61	
					63	$3500			2.5	
					290	€1200			0.74	
					630	$2300	4,380		2.41	
					630	$3500	4,380		2.47	
2012	Szmerekovsky and Venkateshan	MIP	TCT	na	90	na	206	na	na	An MIP method capable of obtaining solutions for TCTPs with irregular cost functions. This method can be adapted for Net Present Value (NPV) minimization of costs and Cash Availability (CA) maximization.

Table 2.1 – Existing exact, heuristic, and meta-heuristic methods for solution of TCTP (*Continued*).

Year	Author(s)	Method	Problem	Platform	# of Acts.	IC	CPU Time (s)	# of Runs	APD (%)	Remarks
2012	Zhang and Ng	ACS	TCT	VBA Macro	18	$500	na	na	0	MAWA is applied to combine the time and cost objectives; ACS-SGPU guarantees diversity while ACS provides quality solutions.
			TCT-PF			$1500			0.004 (Based on 4 Obs.)	
		ACS-SGPU	TCT-PF						0.586 (Based on 5 Obs.)	
2013	Aminbakhsh	SAM-PSO	TCT-DL	C++	18	$200	0.08	10	0	Siemens method is modified and embedded to a custom multi-objective PSO for Pareto oriented optimization of discrete TCTP. Applicability of this approach is not documented for large-scale problems with more than 63 activities.
			TCT		18	$0	0.08		0	
					18	$200	0.08		0	
					18	$500	0.08		0	
					18	$1,500	0.08		0	
					63	$2,300	45		0.02	
					63	$3,500	45		0.02	
			TCT-PF		18	$0	8.18		0.1	
					18	$200	7.98		0.08	
					18	$1,500	7.97		0.03	
					63	$2,300	111.51		0.31	
					63	$3,500	108.02		0.27	
2013	Degirmenci and Azizoglu	B&B	TCT-BT	C#	29 to 35	na	494.88 to 5,094.54	na	na	A mode eliminator and a lower and upper-bound calculator is embedded to a B&B for network reduction purposes.

Table 2.1 – Existing exact, heuristic, and meta-heuristic methods for solution of TCTP (*Continued*).

Year	Author(s)	Method	Problem	Platform	# of Acts.	IC	CPU Time (s)	# of Runs	APD (%)	Remarks
2013	Mungle, Benyoucef, Son, abd Tiwari	FCGA	TCT-PF	MATLAB	7	$0	17.22	50	0.128	Fuzzy clustering technique embeds NSGA-II with crowded-comparison operator together with crowding distance for selection and generation of population.
					12		130.86		na	
					18		847.44		0.531 (Based on 18 Obs.)	
2015	Bilir	MIP	TCT-DL	C#	50	$250	1.03	na	na	MIP approach for large-scale TCTPs which is tested using samples generated by means of ProGen/Max.
					100	$500	15.3			
					200	$500	57.44			
					500	$500	96.39			
					1000	$500	174.04			
			TCT-PF		50	$250	302.67			
					100	$500	260.23			
					200	$500	487.27			
2015	Koo, Hong, and Kim	iMOO (GA)	TCT-PF	OptQuest	7	$500	11	na	na	This GA-based approach is designed for optimization of four fitness functions including trade-offs between time-cost, cost-quality, sustainability-cost, and productivity-safety.
2015	Vanhoucke	Electro-magnetic	TCT-DL	Visual C++	10	na	0.049 to 2.869	na	0.02 to 98.6	This model is used for four different variants of deadline TCTP with Time/switch constraints, work continuity constraints, and NPV maximization considerations. Two local searches are implemented for crashing and relaxing activities.
					20					
					30					
					40					
					50					

Table 2.1 – Existing exact, heuristic, and meta-heuristic methods for solution of TCTP (*Continued*).

Year	Author(s)	Method	Problem	Platform	# of Acts.	IC	CPU Time (s)	# of Runs	APD (%)	Remarks
2016	Aminbakhsh and Sonmez	DPSO	TCT-DL	C++	18	$200	0.04	10	0	A PSO with novel principles for representation, initialization and position-updating of the particles which operates in discrete space and incorporates a semi-deterministic initialization scheme.
			TCT		18	$500	0.04		0	
					18	$1,500	0.04		0	
					63	$2,300	1.3		0.02	
					63	$3,500	1.3		0.05	
					630	$2,300	14.6		0.33	
					630	$3,500	14.6		0.34	
					200	$500	4.6		0.19	
					500	$500	17.2		0.21	
2016	Zou, Fang, Huang, and Zhang	MILP	TCT-DL	na	30	na	1.63 to 44.08	na	na	A MILP and an alternative approximation approach for larger instances is proposed for repetitive problems with fixed logic and multiple crews. This method assumes typical projects with a single mode.
					40		2.55 to 175.43			
					50		3.95 to 546.19			
		TCTP-Approx.			30		0.32		0.06	
					40		0.74		0.04	
					50		1.78		0.06	
					80		56.05		na	
					90		104.42			
					100		643.4			

Table 2.1 – Existing exact, heuristic, and meta-heuristic methods for solution of TCTP (*Continued*).

Year	Author(s)	Method	Problem	Platform	# of Acts.	IC	CPU Time (s)	# of Runs	APD (%)	Remarks
2016	El-Abbasy, Elazouni, and Zayed	MO-SCOP-EA	TCT-PF	na	18	$0	na	na	na	A GA-based model that considers financial and resource aspects. Pure TCTP results of GA is validated only by visual comparison of TCT curves.
2017	Aminbakhsh and Sonmez	PFPSO	TCT-PF	C++	18	$0	2	10	0	PSO is embedded with a simplified-SAM which unlike any previous extension of SAM, finds the least-cost-slope critical activity by considering only a single critical path at a time. This PSO employs the external repository concept and redefines position update and global best calculation routines.
					18	$200	2		0	
					18	$1,500	2		0	
					180	$0	21		0.1	
					360	$0	43		na	
					720	$0	92			
2017	Bettemir and Birgonul	Heuristic	TCT-DL	na	18	$200	na	na	0	Elimination technique of the new Heuristic contributes to faster computations. Paths are considered critical if they do not include critical and non-critical activities simultaneously. A project duration predictor is also employed to reduce the number of schedule evaluations. This is done by only considering the crashed activities and those with floats greater than critical total float amount.
			TCT		18	$200			0	
					63	$2,300			0.046	
					63	$3,500			0.077	

Table 2.1 – Existing exact, heuristic, and meta-heuristic methods for solution of TCTP (*Continued*).

Year	Author(s)	Method	Problem	Platform	# of Acts.	IC	CPU Time (s)	# of Runs	APD (%)	Remarks
2017	Dragović, Vulević, Todosijević, Kostadinov, and Zlatić	LP	TCT-DL	MATLAB	10 to 13	na	na	na	na	An LP model that is used for direct cost minimization of river regulation and check dam construction projects. This model neglects daily indirect cost in the optimization process.
2017	Su, Qi, and Wei	Heuristic	TCT-DL	LINDO	116	na	2	na	na	Equivalent simplification-based approach which transforms nonlinear continuous TCTP into a simple activity float problem. It uses a complex polynomial algorithm to solve the transformed problems.
2018	Agdas, Warne, Osio-Norgaard, and Masters	Parallel-GA	TCT-DL	Python	1800	$200	20,952	5	7.05	A parallel GA which is modified using new techniques including problem encoding, weighted graph, parallel fitness evaluation, and stagnation criteria for solutions of large-scale problems.
			TCT		63	$2,300	111		0.26	
					63	$3,500	88.8		1.24	
					630	$2,300	364.2		2.76	
					630	$3,500	396.6		2.29	
					1800	$1,500	21,024		14.72	
					3150	$2,300	32,940		6.5	
					3150	$3,500	33,876		4.73	
					6300	$2,300	59,112		7.66	
					6300	$3,500	60,336		6.96	

Respecting the state of the existing research on TCTP in the construction industry, it can be observed that GAs are the most prominent methods used in a multitude of studies. Above all, it can be concluded that the majority of the articles address instances with small problem networks and that detailed performance evaluation on accuracy and efficiency of most of the presented approaches appear to be lacking. In fact, performances of the algorithms are often only reported for the small benchmark network of Feng et al. (1997) that includes 18 activities with up to five time-cost alternatives. Kalhor et al. (2011), Fallah-Mehdipour et al. (2012) and Mungle et al. (2013) are among the few studies reporting on different performance metrics of their proposed models. Besides, it is worth mentioning that although a large body of the literature has hitherto been dedicated to development of optimization methods for DTCTP, only some of these methods are used in real-life practices. That is largely resulting from the fact that they do not suit actual practices and that major domain of the literature focus on proving applicability of various optimization models rather than providing means for optimization of real construction projects. To expand on this, very few of the existing methods can be applied to optimization of real-life construction projects which typically comprise more than 300 activities (Liberatore et al., 2001). Furthermore, a few methods that are tested for real-life-size large-scale problems, require enormous computation time and resources thanks to the inherent complexity of solving DTCTPs. The few efforts that handle large-scale problems include parallel GA of Kandil and El-Rayes (2006), constrained programming of Menesi, Golzarpoor, and Hegazy (2013), and parallel GA of Agdas et al. (2018) employing problems including up to 720, 2000, and 6300 activities, respectively. However, all the practiced large-scale problems are based on small networks which are generated by copying the base problem in serial several times; hence, these problems are believed to have limitations in reflecting the complexity of the real-life construction projects. In addition, the few studies practicing large-scale instances require an unreasonably large computation time; for instance, GA-based approach of Kandil and El-Rayes (2006) require 15.4 hours over a supercomputing cluster of 50 processors to solve a 720-activity

problem. Likewise, GA-based approach of Agdas et al. (2018) require 16.7 hours on a high-performance computing facility with eight CPU cores to unravel a 6300-activity problem. Nevertheless, for any solution method to be practically viable, accuracy needs to be accompanied with the efficiency. The main reasons behind the gap between the theoretical achievements of researchers and practical applications of professionals is not only related to the inherent computational complexity of large-scale networks, but also due to the dearth of real-life-scale problems. Therefore, presenting performance of large-scale, realistic TCTP instances is one of the most important contributions of this thesis. In order for better evaluation of the capabilities of the proposed optimization models, new sets of multi-mode large-scale DTCT problems have been generated by means of random project instance generators. Unlike a large body of the existing literature, performance measurements are carried out by taking a more holistic approach that involves a set of performance comparison indices. Hence, the presented findings are highly relevant for real-life applications.

With regard to the summary of the past research given in Table 2.1, it can also be observed that the literature is not rich with the more complex Pareto oriented optimization in the domain of TCTP, especially for the discrete version of this problem. This is despite the fact that obtaining non-dominated sets of solutions is widely acknowledged as the ultimate resolution of TCTP studies (e.g. Zheng et al., 2005; Yang, 2007b; Eshtehardian et al., 2008; Aminbakhsh and Sonmez, 2017). This thesis study emphasizes the importance of discrete and Pareto front TCTPs due to their practical relevance and because they facilitate expressing decision makers preferences so that they can select the best solution based on their own concerns. To the respect of this, new models with exceptional accuracy and efficiency, applicable in real-life-scale projects are designed and developed in this thesis. This research study aims to contribute to both researchers and practitioners by tightening the gap between the literature and the real-world requirements of the projects. The new proposed models include Mixed-Integer Linear Programming

techniques that use Gurobi solver version 6.0.5, new Particle Swarm Optimizers, and new Cost-Slope Heuristics. For all the proposed methods, two variants have been designed and developed to address both the deadline and the Pareto front classes of DTCT problems. Since, exact procedures are the only methods guaranteeing optimality of the solutions and that heuristics and meta-heuristics are incapable of securing the optimality of the solutions, the proposed Mixed-Integer Linear technique is used in performance evaluation of the developed heuristic and meta-heuristic methods.

Last but not least, despite the fact that any scientific decision support tool would have a pivotal role in the decision-making process, none of the commercial scheduling software packages (e.g., Microsoft Project, Primavera) include tools or modules for TCT analyses of the scheduling problems. Therefore, this thesis study also presents integration of DTCTP optimization modules into Microsoft Project – a widely used commercial planning software in the construction industry – by means of an add-in which is capable of solving two variants of DTCTP, namely, cost minimization/deadline and Pareto problems. The integrated modules include both the proposed Particle Swarm Optimizer and the Cost-Slope Heuristic. By means of this, the new models are envisioned to be applicable in real projects and to suit the actual practices of construction managers. In the ensuing chapters, characteristics of the proposed particle swarm optimizers, Cost-Slope Heuristics, and Mixed-Integer Linear Programming methods developed for solution of different extensions of discrete time-cost trade-off problems are going to be presented. Meanwhile, since the approaches presented in this thesis are non-domain-specific, they can easily be used for solution of similar optimization problems.

CHAPTER 3

DISCRETE PARTICLE SWARM OPTIMIZATION METHOD FOR DTCTP

As is clear, there is a lack of Particle Swarm Optimization (PSO) exemplars in the existing literature with the capacity to tackle realistic large-scale construction DTCT problems efficiently and effectively to make them practicable in real-life projects. Accordingly, this chapter includes the background and theoretical properties of PSO method. This chapter also covers a background of Siemens Approximation Method (SAM), a modified variant of which is used for development of a new PSO model. Following the definitions and principles of the base models, the proposed discrete particle swarm optimization method for cost minimization and deadline extensions of DTCTP is described. Results of inclusive comparative studies are presented which prove the outstanding performance of the proposed approach with solid convergence capabilities.

3.1. Particle Swarm Optimization (PSO)

Studies on biological evolution and collective behavior extant in natural systems such as animal herds, schools of fish, and flocks of bird established the primitive initiatives for development of methods based on swarm intelligence. The earliest precursor for swarm intelligence-based optimization approach encompass the first paradigm of the PSO developed by Kennedy and Eberhart (Eberhart and Kennedy, 1995; Kennedy and Eberhart, 1995) who later introduced the binary version of this algorithm for problems with discrete search spaces (Kennedy and Eberhart, 1997).

PSO is rooted upon imitating the choreography of bird flocks that communicate together as they fly; therefore, the population is called “swarm”, while, the potential solutions are named as “particles”. This algorithm conceptually resembles evolutionary algorithms and vastly relies on stochastic procedures.

The system initializes with a population of random potential solutions. Particles iteratively fly over the search space in explicit directions and are attracted to self-attained historical best position (personal best; *pbest*), as well as the best position among the entire swarm (global best; *gbest*). Each particle records the coordinates associated with the best location it has visited so far. At each time step, particles evaluate their own positions with respect to fitness criteria, then by comparing the fitness values, they communicate to identify the particle located at the best position. Each particle moves towards the best position using a velocity that incorporates coordination of the personal best location as well; then, it evaluates the domain from its new location, and the process reiterates until either the swarm reaches to a predefined target, or a computational limit.

3.2. Siemens Approximation Method (SAM)

Siemens approximation method (SAM) (Siemens, 1971) is one of the first heuristics developed for TCTP. SAM involves a logical and systematic procedure to minimize the overall project cost. It was originally developed for the continuous TCT problem and is capable of obtaining solution for convex nonlinear TCTPs by making multiple piecewise linear curve approximations. The procedure initiates with the construction of the project network, thereby, crashing critical activities one at a time based on some specific rules. Of these rules, the most important is selection of a critical activity with minimum amount of cost-slope. The act of crashing continues until either all the activities are crashed, or those with normal duration have cost-slopes greater than the daily indirect cost rate. Nevertheless, SAM accelerates the project based on the minimum cost-slope calculated at each iteration with respect

to the all-normal (uncrashed) and all crashed options. However, in discrete problems, it is possible to have more than two options for each of the activities. Thus, $m-1$ number of cost-slopes can be calculated for an activity with m number of alternatives. This study extends the use of SAM to discrete TCT problems by modifying some of the major steps of this heuristic method. This modified variant, hereafter called modified-SAM, is later embedded into the discrete PSO in order to improve the quality of the initial swarm. The new SAM-based heuristic – similar to the original method – is initiated with the construction of the project network using the normal modes. The activity with minimum cost-slope is identified and crashed according to Eq. (3.1). In case of a tie, the activity leading to a shorter project duration is selected, if the tie is not broken, the activity with the smaller activity number is selected.

$$CS_j = (dc_{jk} - dc_{j(k-1)})(d_{jk} - d_{j(k-1)})^{-1}$$
$$\forall\, j = \{1,...,S\} \quad , \quad \forall\, k = \{1,...,m(j)\} \tag{3.1}$$

where; S is the number of activities, $m(j)$ is the number of available time-cost options for activity j, dc_{jk} is the direct cost of k th time-cost option of activity j, d_{jk} is the duration of the k th time-cost option of activity j. The cost-slopes of the first network are evaluated by setting $k = m(j)$ initially; and then, decreasing the value of option k, one at a time, as the j th activity is crashed.

3.3. Discrete Particle Swarm Optimization Method (DPSO)

Few researches have presented PSO algorithms for the DTCTP (Yang, 2007a; Bettemir, 2009; Zhang and Xing, 2010; Fallah-Mehdipour et al., 2012) which have operated in continuous space. In this thesis, a discrete particle swarm optimization (DPSO) algorithm is designed to achieve an improved particle swarm representation for cost minimization and deadline DTCTPs. Previous researches

employ random initialization schemes to produce the first generation of particles. However, the proposed DPSO method is equipped with a semi-random pattern for generation of the initial particles. To this end, DPSO is designed to create a certain percentage, pct, of the initial population of size N by using the modified-SAM method and generating the rest of the swarm randomly. The solutions obtained by using modified-SAM are fed into the particle swarm optimizer of DPSO. Positions for the remaining $1-pct$ percent of the population are set randomly. That is, for an S-dimensional problem, PSO generates S number of random values for each variable j which are selected from the feasible range $\{1, m(j)\}$. In the case of TCTP, S resembles the number of activities and the feasible range $\{1, m(j)\}$ represents the available time-cost alternatives for the jth activity. Solutions are denoted by x_{ijk} which represents a binary value that holds ith particle's position for the jth activity, which can only have one k value equal to one while all the remaining positions for the jth activity of the same particle are set to zero ($\sum_{k=1}^{m(j)} x_{jk}^{(t)} = 1$). Logical sequencing of the activities for the generated particles are implemented according to Eq. (3.2).

$$ES_j^{(t)} + d_j^{(t)} - ES_l^{(t)} \le 0 \quad , \quad \forall\, l \in Sc_j \tag{3.2}$$

where; t is the generation number; d_j is activity j's duration; ES_j is the early start time of jth activity; and Sc_j are the immediate successors of the jth activity. In the first iteration, bound by the feasible region $[-v_{\max}, v_{\max}]$, random velocity vectors, $v_{ijk}^{(1)}$, are determined for all the initial seeds (i.e., deterministic and random particles). Velocities of the individuals are clamped to avoid swarm divergence. Generation of the initial particles is completed with the determination of the $pbest$, P_i^{t0} and the $gbest$, P_g^{t0} positions. The indices of the particle's best and best

particle among the entire swarm are represented by i and g, respectively. Each particle acquires the "best" positions, using Eq. (3.3) as follows:

$$P_{ijk}^{t0} = P_{gjk}^{t0} = x_{ijk}^{(1)} \tag{3.3}$$

Each particle i's fitness is evaluated with respect to the Eq. (3.4) and Eq. (3.5), which minimize the sum of direct and indirect costs. Incentive and disincentive payments are explicitly considered in project cost calculations.

$$D_i = \max\left[\sum_{j=1}^{S}\sum_{k=1}^{m} d_{jk}^{(t)} x_{jk}^{(t)}\right] \tag{3.4}$$

$$\begin{aligned} C_i &= \begin{cases} \sum_{j=1}^{S}\sum_{k=1}^{m(j)} dc_{jk}^{(t)} x_{jk}^{(t)} + D_i \times ic + T \times dp & \text{if } T \geq 0 \\ \sum_{j=1}^{S}\sum_{k=1}^{m(j)} dc_{jk}^{(t)} x_{jk}^{(t)} + D_i \times ic - T \times db & \text{otherwise} \end{cases} \\ \forall j &= \{1,\ldots,S\} \quad , \quad \forall k = \{1,\ldots,m(j)\} \end{aligned} \tag{3.5}$$

where; d_{jk} and dc_{jk} represent duration and direct cost of the k th options of the j th activity, respectively; ic denotes the daily indirect cost rate; D_i and C_i represent duration and cost of the project for the i th particle; D_{dl} is the project deadline; dp is the daily delay penalty; db denotes the daily bonus.

The qualities of the solutions are compared with each other according to Eq. (3.6):

$$u \succ v \quad \text{if} \quad C_u < C_v \tag{3.6}$$

which makes the discrimination in favor of decision vector u in case the total cost of that particle is less than decision vector v; while, in case of equality ($C_u = C_v$) discrimination is made in favor of the particle with smaller duration by Eq. (3.7).

$$u \succ v \quad if \quad \begin{cases} C_u = C_v \\ D_u < D_v \end{cases} \tag{3.7}$$

For the occasion in which both particles u and v have the same total costs and durations, discrimination is made randomly. P_i's and P_g's are updated after identification of the better fitted individuals. Particles are flown to their new positions using the velocity vector given in Eq. (3.8), which incorporates Kennedy and Eberhart's (1997) linearly decreasing time-varying inertia weight (w) (Eq. (3.9)). This parameter provides the PSO with more exploration ability at the initial stages followed by more exploitation ability at the closing cycles. According to Eq. (3.8), each particle updates its velocity using current velocity, the distance to personal best experience, and the distance to best position of the entire swarm.

$$v_{ijk}^{(t+1)} = w^{(t)} v_{ijk}^{(t)} + c_1 r_1 \left(P_{ijk}^{(t)} - x_{ijk}^{(t)} \right) + c_2 r_2 \left(P_{gjk}^{(t)} - x_{ijk}^{(t)} \right) \tag{3.8}$$

$$w = w_{min} + \left(w_{max} - w_{min} \right) \left(\frac{t_{max} - t}{t_{max} - 1} \right) \tag{3.9}$$

where; r_1 and r_2 are random vectors with uniformly distributed components within the range [0,1]; the constants c_1 and c_2 are the cognitive and social parameters, respectively; $w_{\min}$ and $w_{\max}$ denote lower and upper-bounds of inertia weight, respectively; and $t_{\max}$ is the total number of iterations.

The values of c_1 and c_2 parameters control the convergence capabilities of a particle by biasing its movement toward *pbest* or *gbest* positions, respectively.

Velocity vectors are transformed into probabilities (Aminbakhsh, 2013) and are normalized to the range [0,1] using a logistic transformation function given in Eq. (3.10). Each particle is then migrated to a new position subject to the probabilistic condition according to Eq. (3.11).

$$sig\left(v_{ijk}^{(t)}\right) = \frac{1}{1+\exp\left(-v_{ijk}^{(t)}\right)} \tag{3.10}$$

$$x_{ijk}^{(t+1)} = \begin{cases} 1 & if \quad sig(v_{ijk}^{(t+1)}) = \max\left\{sig(v_{ijk}^{(t+1)})\right\} \\ 0 & otherwise \end{cases} \tag{3.11}$$

where each $sig(v_{ijk}^{(t)})$ represents the probability that $x_{ijk}^{(t+1)}$ would be selected.

Eq. (3.11) differs from the position update equation of the binary PSO proposed by Kennedy and Eberhart (1997), in that, it involves determination of the alternative(s) associated with the maximum amount of probability for every activity. This condition indicates that in each row of position matrix, a single alternative with the largest probability will have a value of one. If the value of $\max\left\{sig(v_{ijk}^{(t+1)})\right\}$ is same for more than one alternative, then, discrimination is made randomly.

3.3.1. Case Example

Given in Figure 3.1, a hypothetical case example including six non-dummy and two dummy activities is introduced herein which will be used to elucidate various concepts, definitions, and applicability of the approaches that are proposed within the context of this thesis study. This case example is used in this section to illustrate the binary representation for particles' positions.

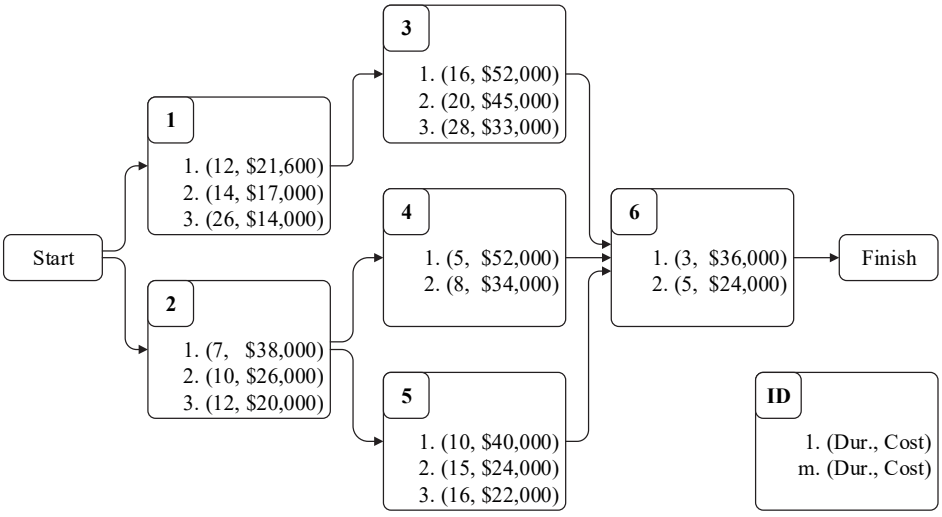


Figure 3.1 – Case Example.

Figure 3.2 illustrates the probability matrix for the 5th particle in the 2nd iteration for the case example. The velocity matrix $v_5^{(2)}$, which is calculated by Eq. (3.8), is transformed to the probability matrix $sig\left(v_5^{(2)}\right)$ using the sigmoid function given in Eq. (3.10).

	Modes		
Activities	k_1	k_2	k_3
j_1	0.71	0.11	0.73
j_2	0.94	0.94	0.97
j_3	0.82	0.12	0.27
j_4	0.75	0.96	
j_5	0.28	0.97	0.31
j_6	0.69	0.51	

Figure 3.2 – Probability matrix.

The 5th particle's new position matrix ($x_5^{(3)}$) in the 3rd iteration is determined by using the Eq. (3.11) as shown in Figure 3.3.

Activities	Modes k_1	k_2	k_3
j_1	0	0	1
j_2	0	0	1
j_3	1	0	0
j_4	0	1	
j_5	0	1	0
j_6	1	0	

Figure 3.3 – Position matrix.

The discrete position matrix in Figure 3.3, assigns a value of "one" to the mode selected, and a value of "zero" to the remaining modes, to make a clear discrimination of the selected modes. For example, for the first activity the third mode is selected, hence the position x_{13} is assigned a value of "one", and the remaining positions of the first activity (x_{11} and x_{12}) are assigned a value of "zero". If a continuous position matrix was used instead, the probability of x_{11} (0.71) would be very close to that of x_{13} (0.73), hence a clear discrimination between the first mode and the third mode would not be made as the two probabilities are very close to each other.

The optimization process is reiterated until the preset number of iterations is reached. DPSO returns the final *gbest* particle as the solution for the DTCT problem when the optimization process is terminated. The proposed DPSO method is graphically explained as a flowchart in Figure 3.4 and the pseudo-code of this algorithm is illustrated in Figure 3.5.

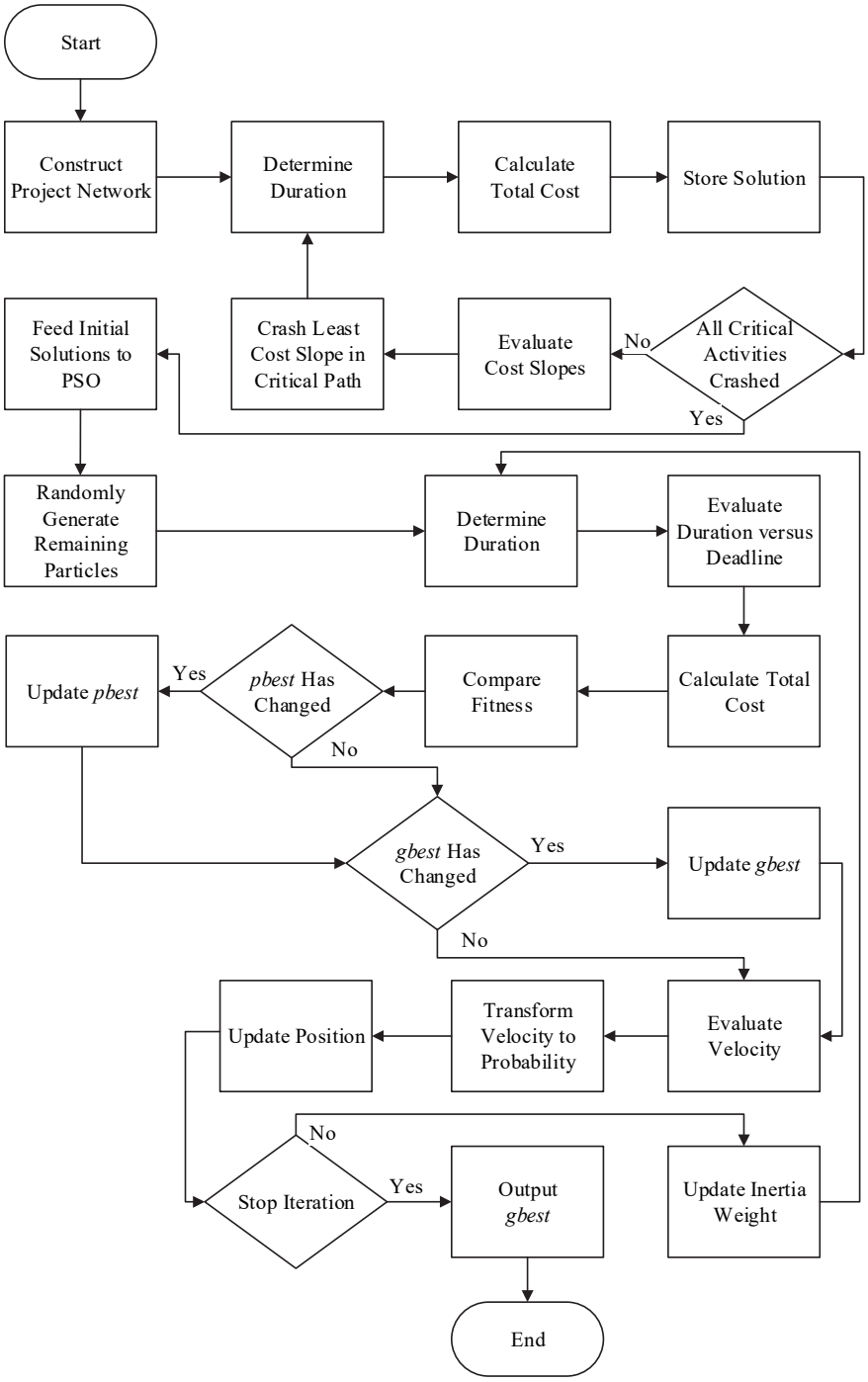


Figure 3.4 – Flowchart of the proposed discrete PSO algorithm.

```
Begin
    For ∀ j ∈ [1, S]
        For ∀ k ∈ [1, m]
            Retrieve Values;
        End;
    End;
    Construct network;
    While ∃ j ∈ Critical-Acts: Fully-Crashed ≠ True ∧ i ≤ pct.N
        Calculate CPM;
        Calculate Dur/Cost;
        Store Solution;
        Calculate CS;
        Crash Min-CS;
        Set x_i as pbest and gbest;
        Set random Velocity;
    Break;
    For ∀ i ∈ (pct.N, N]
        For ∀ j ∈ Acts
            Set random Position;
            Set random Velocity;
            Set x_i as pbest and gbest;
        End;
    End;
    While t ≤ t_max
        For ∀ i ∈ [1, N]
            Calculate CPM;
            Calculate Dur/Cost;
            For ∀ j ∈ Acts
                Set w;
                If x_i ≻ P_i
                    Set x_i as pbest;
                    If P_i ≻ P_g
                        Set P_i as gbest;
                    End;
                End;
                Calculate Velocity;
                Transform Velocity to Probability;
                Update Position;
            End;
            Update w;
        End;
    Break;
    Return gbest;
End;
```

Figure 3.5 – Pseudo-code of the proposed discrete PSO algorithm.

3.4. Computational Experiments of DPSO

Computational experiments are conducted to evaluate the performance of the proposed DPSO method for the DTCTP using a set of benchmark instances acquired from the literature as well as a set of large-scale problems introduced in this section. The proposed optimization algorithm is coded in C++ and compiled within Visual Studio 2013 on a 64-bit platform. All of the tests are carried out on a desktop computer with a P9X79 Chipset motherboard, 16 GB 667 MHz DDR3 random-access memory (RAM), Intel Core i7-3.40 GHz CPU, and 64-bit Windows 10 operating system. DPSO is executed solely (no other software is ran simultaneously) on a single processor and overclocking is not performed.

3.4.1. Parameter Configuration of DPSO

It is broadly acknowledged that evolutionary algorithms are very sensitive to configuration of their parameters and the proposed DPSO is not an exception to this. Hence, pilot experiments were conducted to determine an adequate set of parameter values for the DPSO. The pilot experiments revealed that the set of parameters that are summarized in Table 3.1 provided an adequate combination for the DPSO.

Table 3.1 – Parameter configuration of the DPSO.

Parameter	Description	Value
i	# of Birds	250
pct	% of deterministic swarm	0.8
c_1	Cognitive Parameter	5
c_2	Social Parameter	1
$w_{\max}$	Max. Inertia Weight	1.2
$w_{\min}$	Min. Inertia Weight	0.0
$v_{\max}$	Max. Velocity	3.7

50,000 schedules (objective function evaluations) is used as the stopping criterion in all of the experiments (Kolisch and Hartmann, 2006; Sonmez and Bettemir, 2012). Since PSO is a stochastic search algorithm, performance of DPSO is evaluated for ten consecutive runs for each instance, and average percent deviation (APD) from the global optimal solution is reported.

3.4.2. Small-Scale Benchmark Problems

The performance of the proposed DPSO method was first tested using the small-scale DTCTP test instances which are commonly used in the literature (Elbeltagi et al., 2005; Zheng et al., 2005; El-Rayes and Kandil, 2005; Kandil and El-Rayes, 2006; Elbeltagi et al., 2007; Ng and Zhang, 2008; Xiong and Kuang, 2008; Afshar et al., 2009; Fallah-Mehdipour et al., 2012; Sonmez and Bettemir, 2012; Zhang and Ng, 2012; Monghasemi, Nikoo, Khaksar Fasaee, and Adamowski, 2015) for performance evaluations. The small-scale test instances consisted of an 18-activity network (Feng et al., 1997) with the time-cost alternatives defined in Hegazy (1999). Although it has not been pointed out by any other preceding study, it is worth mentioning that this problem is flawed since the cost of third time-cost alternative of the eighth activity must have been selected from the interval $\mathrm{DU}(208,215)$; however, the assigned cost is \$200. Nevertheless, in order to conduct a fair comparison with the previous studies, the benchmark problem is used without applying any corrections. The activity on node (AoN) representation of this instance can be obtained from Feng et al. (1997) and the time-cost data can be attained from Hegazy (1999). This problem includes one activity with two modes, ten activities having three modes, two activities with four modes, and five activities with five modes; accounting for a total of 5.9×10^{9} possible schedules. This benchmark problem is examined under three different conditions. In problem 18a, the indirect cost figure is \$200/day, the delay penalty is set as \$20,000/day, the incentive payment is assumed as \$1,000/day, and the completion deadline is assigned as 110 days. In problem 18b, the indirect cost rate is \$1,500/day, and in

problem 18c, the indirect cost is $500/day. The optimal solutions for the practiced small-scale problems are determined by applying the mixed-integer programming formulation given in Eqs. (1.1)-(1.6) using AIMMS 4.2 optimization software. The optimal results are also verified by means of a mixed-integer linear programming technique that employs Gurobi solver 6.0.5. This method is discussed in detail in Section 5.2.3. The optimal solutions for problems 18a, 18b, and 18c are calculated as $128,270, $271,270, and $161,270. Snapshots of the performance of the results for problems 18a, 18b, and 18c are given in Table 3.2, Table 3.3, and Table 3.4, respectively. The average percent deviations from global optima of DPSO compare favorably with the existing methods which are summarized in the following tables.

Table 3.2 – Performance of DPSO for problem 18a.

Algorithm	**# of runs**	**APD (%)**
GA (Hegazy, 1999)	1	8.139
GA (Sonmez and Bettemir, 2012)	10	2.17
HA (Sonmez and Bettemir, 2012)	10	0.00
DPSO (Section 3.3)	10	0.00

Table 3.3 – Performance of DPSO for problem 18b.

Algorithm	**# of runs**	**APD (%)**
MAWA-GA (Zheng et al., 2005)	1	0.903
ACS-TCO (Ng and Zhang, 2008)	1	0.018
NA-ACO (Afshar et al., 2009)	1	0.00
ACS-SGPU (Zhang and Ng, 2012)	1	0.698
ACS (Zhang and Ng, 2012)	1	0.018
GA (Sonmez and Bettemir, 2012)	10	1.29
HA (Sonmez and Bettemir, 2012)	10	0.00
DPSO (Section 3.3)	10	0.00

Table 3.4 – Performance of DPSO for problem 18c.

Algorithm	# of runs	APD (%)
GA (Elbeltagi et al., 2005)	20	2.171
MA (Elbeltagi et al., 2005)	20	0.759
PSO (Elbeltagi et al., 2005)	20	0.415
ACO (Elbeltagi et al., 2005)	20	3.351
SFL (Elbeltagi et al., 2005)	20	2.960
MSFL (Elbeltagi et al., 2007)	20	0.00
ACS-TCO (Ng and Zhang, 2008)	1	0.00
Electimize (Abdel-Raheem and Khalafallah, 2011)	20	0.00
ACS-SGPU (Zhang and Ng, 2012)	1	0.00
ACS (Zhang and Ng, 2012)	1	0.00
DPSO (Section 3.3)	10	0.00

In all of the ten trials for the three problems, DPSO was able to obtain the global optimal results within 50,000 schedules by searching only 8.47×10^{-4} percent of the entire solution space. As a result, DPSO was able to solve all the small-scale test instances in 0.4 seconds on average.

The comparison of DPSO with the state-of-the-art methods proves that proposed DPSO is among the top performing algorithms for the small-scale DTCTP. The DPSO has outperformed the genetic algorithms (Hegazy, 1999; Sonmez and Bettemir, 2012) for instance 18a (Table 3.2). Similarly, for instance 18b, DPSO was better than the genetic algorithms (Zheng et al., 2005; Sonmez and Bettemir, 2012), ant colony system time-cost optimization algorithm (ACS-TCO) of Ng and Zhang (2008), and ant colony system (ACS) and ant colony system with global updating strategy (ACS-SGPU) algorithms of Zhang and Ng (2012) for obtaining the optimal solution. The proposed DPSO also outperformed the genetic, memetic, PSO, ant colony optimization (ACO), and shuffled frog-leaping optimization (SFL) algorithms of Elbeltagi et al. (2005) for the problem 18c.

The modified shuffled frog-leaping optimization algorithm (MSFL) (Elbeltagi et al., 2007), the nondominated archiving ACO (NA-ACO) (Afshar et al., 2009), the

Electimize algorithm (Abdel-Raheem and Khalafallah, 2011), and the hybrid genetic algorithm with simulated annealing (HA) (Sonmez and Bettemir, 2012) are among the state-of-the-art methods that are competitive with the proposed DPSO for obtaining high quality solutions for the small instances having no more than 18 activities. Though, the elapsed CPU time for MSFL was eight seconds compared with 0.4 seconds of the DPSO. Besides, except for GA and HA (Sonmez and Bettemir, 2012), none of the existing state-of-the-art methods were tested for larger instances.

3.4.3. Medium-Scale Benchmark Problems

The medium-scale instances used for performance evaluation of DPSO consisted of the 63-activity problem of Sonmez and Bettemir (2012). The activity on node (AoN) topological representation and the time-cost data of this instance can be obtained from Sonmez and Bettemir (2012). This benchmark problem contains two activities with three modes, fifteen activities with four modes, and forty-six activities having five modes, totaling 1.37×10^{42} different schedules. This test instance is fitted into the model by assuming two indirect cost figures. In the first problem, 63a, the indirect cost is set as \$2,300/day, and in the second problem, 63b, the indirect expense is assumed as \$3,500/day. The optimal solutions for the practiced medium-scale problems are determined and verified by the methods pointed out in Section 3.4.2. The MIP/AIMMS and the MILP/Gurobi approaches provided the optimal solutions for problems 63a and 63b as \$5,421,120 and \$6,176,170, respectively. By evaluating only 3.64×10^{-36} percent of the solution space, DPSO achieved APD values of 0.02% and 0.05% for instances 63a and 63b, respectively. The proposed DPSO algorithm was very successful for obtaining high quality solutions for the medium-scale test instances and outperformed the sole-genetic algorithm and hybrid-genetic algorithm (Sonmez and Bettemir, 2012) as shown in Table 3.5.

Table 3.5 – Performance of DPSO for problems 63a and 63b.

Algorithm	63a		63b	
	# of runs	APD (%)	# of runs	APD (%)
GA (Sonmez and Bettemir, 2012)	10	5.86	10	5.16
HA (Sonmez and Bettemir, 2012)	10	2.61	10	2.50
DPSO (Section 3.3)	10	0.02	10	0.05

The contents of Table 3.5 shed some light on exceptional performance of DPSO for solution of medium-scale problems. By searching the same amount of 50,000 solutions, HA was able to achieve APD values of 2.61% and 2.50%. DPSO was able to determine very high-quality solutions, with very marginal deviations from the optimal in just 1.3 seconds on average.

3.4.4. Large-Scale Benchmark Problems

The large-scale problem used for performance measurement of the proposed DPSO comprise 630 activities. In fact, this problem is generated by copying the 63-activity problem of Sonmez and Bettemir (2012) ten times in serial. The approach of creating multiple copies of the base problem has the benefit of knowing the expected optimal solution. This instance incorporates 20 activities with three modes, 150 activities with four modes, and 460 activities having five modes, totaling 2.38×10^{421} possible realizations. This instance represents the size of a realistic construction project and is studied under two assumptions regarding the amount of the daily indirect cost. In the first problem, 630a, the indirect cost is set as \$2,300/day, and in the second problem, 630b, the indirect expense is assumed to be \$3,500/day. The optimal solutions for the practiced large-scale problems are calculated as the multiples of the optimal solutions of 63-activity instance used in Section 3.4.3. Accordingly, the optimal solutions for problems 630a and 630b are determined as \$54,211,200 and \$61,761,700, respectively. The performance of the DPSO for large-scale instances is summarized in Table 3.6. DPSO achieved very

successful results and outperformed the hybrid genetic algorithm (HA) (Sonmez and Bettemir, 2012) for large-scale instances.

Table 3.6 – Performance of DPSO for problems 630a and 630b.

Algorithm	630a		630b	
	# of runs	APD (%)	# of runs	APD (%)
GA (Sonmez and Bettemir, 2012)	10	8.83	10	7.50
HA (Sonmez and Bettemir, 2012)	10	2.41	10	2.47
DPSO (Section 3.3)	10	0.33	10	0.34

The APD values for instances 630a and 630b were 0.33% and 0.34%, respectively; and the processing time of DPSO was 14.6 seconds on average. By searching only 50,000 solutions out of 2.38×10^{421} potential solutions, DPSO was able to achieve high quality solutions for the large-scale instances. HA was able to provide solutions with APD values of 2.41% and 2.47% by evaluating the same amount of 50,000 schedules. It was also revealed by Bettemir (2009) that HA was able to achieve an APD value of 2% by solving the 630-activity problem in 73 minutes through 1,000,000 schedule evaluations. Compared with HA, DPSO was able to achieve significantly better solutions in a much shorter computational time.

3.4.5. New Sets of Instances

As is clear, the network and the data for the large-scale benchmark problem included in the literature are generated by copying a simpler problem several times that might not fully reflect the complexity of some real-life-scale construction projects. To this end, 80 new large-scale instances with more complex activity networks are generated to further evaluate the performance of the proposed DPSO. ProGen/Max network generator developed by Schwindt (1995) (Figure 3.6) is used to generate the networks for the new sets of problems.

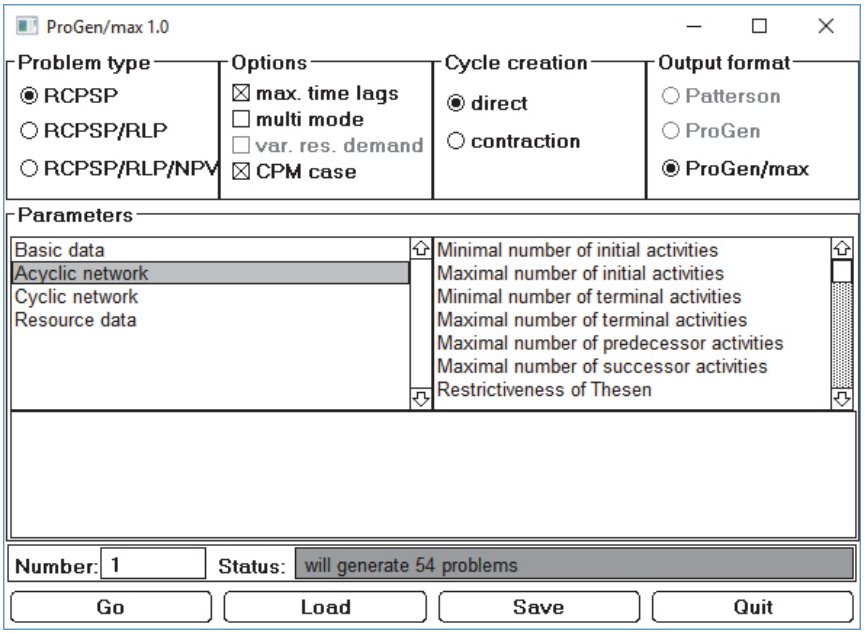


Figure 3.6 – ProGen/Max Interface.

Project networks are developed using four different complexity indices which are represented by Thesen restrictiveness coefficient in ProGen/Max. For each Thesen restrictiveness coefficient value of 0.2, 0.4, 0.6, and 0.8, ten networks are generated totaling 40 networks of size 200 and 40 networks with 500 activities. The parameters of ProGen/Max are configured so as to let "Minimal number of Initial acivities" and "Minimal number of terminal activities" be one and "Maximal number of initial activities", "Maximal number of terminal activities", "Maximal number of predecessors", and "Maximal number of successors" be 20. Microsoft Excel is used to generate time-cost alternatives for each network. First, the available number of time-cost alternatives is decided by randomly selecting a value from the interval $DU(2,5)$ (i.e., discrete uniform distribution with parameters 2 and 5) for each of the activities. Thereafter, using the procedure described in Akkan, Drexl, and Kimms (2005), duration of each time-cost alternative is selected from the range obtained by dividing the interval $DU(3,123)$ into the number of alternatives. Cost of time-cost alternatives are generated using convex cost functions as proposed by Akkan et al. (2005). Cost for the normal (uncrashed) mode is randomly selected

from the range $DU(100,50000)$. Cost of the remaining alternatives are calculated using Eq. (3.12). In addition, the indirect cost rate is set as \$500/day for both the networks sizes.

$$dc_{jk} = dc_{j(k-1)} + CS_{jk} \times (d_{j(k-1)} - d_{jk})$$
$$\forall\, j = \{1,...,S\} \quad , \quad \forall\, k = \{2,...,m(j)\} \tag{3.12}$$

where dc_j is the direct cost of the k th alternative for the j th activity; CS_{jk} is the cost-slope which is generated randomly between the range $U(0.1,1)$; d_{jk} is the duration of the k th options of the j th activity.

The new instances were solved to optimality by means of the mixed integer programming model presented in Eqs. (1.1)-(1.6) and using Gurobi solver version 6.0.5; this method will later be discussed in detail in Section 5.2.3. Optimal solutions were achieved for all the 40 instances with 200 activities, and for 36 of the instances with 500 activities within a CPU time limit of five hours for each. The results of the proposed DPSO at the end of the 50,000 schedules are compared with the optimal solutions for the 76 instances that are solved by the mixed integer programming method. As shown in Table 3.7, DPSO achieved an average 0.19% deviation from the optimal solutions within an average CPU time of 4.61 seconds for 40 instances with 200 activities. As illustrated in Table 3.8, it also succeeded an average 0.21% deviation from the optimal solutions within an average CPU time of 17.24 seconds for 36 instances with 500 activities. Unavailable values are tabulated as ‘na’ in Table 3.8.

Table 3.7 – Performance of DPSO for 200-activity instances.

Problem	Optimal		DPSO			
	Dur. (day)	Cost ($)	Dur. (day)	Cost ($)	CPU Time (s)	APD (%)
T200_1	1120	6,379,332	1141	6,393,871	3.58	0.23
T200_2	1006	5,809,665	875	5,847,638	3.58	0.65
T200_3	1534	6,124,407	1534	6,128,020	4.00	0.06
T200_4	1064	5,957,758	1066	5,979,991	3.75	0.37
T200_5	1264	5,726,445	1264	5,730,567	3.59	0.07
T200_6	1085	6,182,999	1087	6,184,392	3.61	0.02
T200_7	1316	6,430,572	1316	6,446,318	3.88	0.24
T200_8	1008	5,812,144	1005	5,832,495	3.72	0.35
T200_9	1186	5,863,689	1190	5,878,431	3.84	0.25
T200_10	1341	6,006,200	1344	6,020,848	3.66	0.24
T200_11	1690	6,364,254	1686	6,368,641	4.25	0.07
T200_12	1965	6,728,040	1966	6,734,137	4.16	0.09
T200_13	1744	5,955,149	1737	5,978,204	4.61	0.39
T200_14	2143	6,567,311	2143	6,569,344	4.81	0.03
T200_15	1420	5,939,672	1420	5,952,793	4.27	0.22
T200_16	2121	6,996,398	2121	7,030,999	4.80	0.49
T200_17	1534	6,075,423	1543	6,085,157	4.13	0.16
T200_18	1484	6,271,532	1484	6,285,156	4.20	0.22
T200_19	1398	6,114,428	1398	6,127,591	4.33	0.22
T200_20	1421	6,476,044	1422	6,481,519	4.41	0.08
T200_21	2194	6,671,598	2194	6,677,859	5.09	0.09
T200_22	1494	6,258,513	1519	6,281,031	4.81	0.36
T200_23	1625	6,422,509	1626	6,430,370	4.55	0.12
T200_24	2110	6,047,807	2110	6,056,395	4.97	0.14
T200_25	2170	6,544,127	2171	6,546,307	4.97	0.03
T200_26	2114	6,908,248	2114	6,915,186	5.24	0.10
T200_27	1988	6,728,935	1987	6,745,455	4.86	0.25
T200_28	1668	6,404,964	1668	6,429,866	4.70	0.39
T200_29	1849	6,792,867	1850	6,806,045	4.63	0.19
T200_30	1302	6,159,162	1292	6,178,588	4.55	0.32
T200_31	2139	6,440,172	2121	6,445,376	5.24	0.08
T200_32	1913	6,361,151	1913	6,372,306	5.44	0.18
T200_33	1733	6,254,230	1731	6,271,461	5.33	0.28
T200_34	1820	6,541,587	1810	6,547,852	5.39	0.10
T200_35	2446	6,966,652	2450	6,974,364	5.53	0.11
T200_36	1496	6,400,397	1496	6,408,169	5.28	0.12
T200_37	3235	7,404,801	3235	7,405,375	6.36	0.01
T200_38	2009	6,496,720	1999	6,507,846	5.27	0.17
T200_39	2380	6,504,650	2394	6,509,194	5.44	0.07
T200_40	2604	6,396,572	2599	6,399,542	5.69	0.05
Total					**4.61**	**0.19**

Table 3.8 – Performance of DPSO for 500-activity instances.

Problem	Optimal		DPSO			
	Dur. (day)	Cost ($)	Dur. (day)	Cost ($)	CPU Time (s)	APD (%)
T500_1	3155	14,960,582	3132	14,997,226	12.68	0.24
T500_2	2601	15,204,223	2597	15,234,850	11.39	0.20
T500_3	2586	15,449,875	2518	15,477,452	11.97	0.18
T500_4	2328	15,365,411	2255	15,410,429	11.39	0.29
T500_5	2774	15,120,770	2676	15,189,132	12.01	0.45
T500_6	na	na	1934	15,401,892	11.45	na
T500_7	2999	15,524,026	3018	15,572,852	12.27	0.31
T500_8	2858	15,599,044	2870	15,642,713	11.99	0.28
T500_9	2910	15,059,136	2932	15,109,956	11.97	0.34
T500_10	2990	15,568,991	2990	15,601,912	12.48	0.21
T500_11	3763	15,726,437	3768	15,751,034	15.49	0.16
T500_12	2962	14,609,048	2962	14,686,697	13.93	0.53
T500_13	3968	15,234,093	3968	15,259,241	15.75	0.17
T500_14	3102	15,644,352	3094	15,689,067	14.66	0.29
T500_15	4425	16,040,457	4424	16,058,224	16.13	0.11
T500_16	5248	16,037,675	5252	16,052,318	16.84	0.09
T500_17	3140	16,250,370	3142	16,282,414	14.55	0.20
T500_18	3710	15,189,213	3709	15,207,328	15.60	0.12
T500_19	4017	15,707,877	4017	15,746,844	15.87	0.25
T500_20	3655	15,544,444	3657	15,556,715	15.49	0.08
T500_21	na	na	2963	15,613,104	16.91	na
T500_22	3396	15,789,889	3406	15,830,346	17.26	0.26
T500_23	4633	16,926,601	4651	16,959,989	18.66	0.20
T500_24	5014	16,267,849	5014	16,330,184	18.94	0.38
T500_25	6379	18,519,422	6379	18,555,704	21.52	0.20
T500_26	5135	16,939,792	5100	16,972,161	19.72	0.19
T500_27	3521	16,046,011	3467	16,102,094	17.13	0.35
T500_28	3498	15,170,906	3502	15,208,054	16.94	0.24
T500_29	3087	15,361,432	2994	15,432,948	16.50	0.47
T500_30	na	na	3050	15,074,188	15.97	na
T500_31	6518	18,446,098	6518	18,462,904	24.13	0.09
T500_32	8273	17,863,876	8273	17,864,920	25.69	0.01
T500_33	na	na	3210	15,519,551	18.48	na
T500_34	6592	17,611,269	6588	17,641,231	21.67	0.17
T500_35	5643	16,910,284	5641	16,929,501	21.20	0.11
T500_36	4606	16,475,275	4603	16,486,769	20.08	0.07
T500_37	8824	19,047,488	8824	19,052,000	26.27	0.02
T500_38	7101	17,258,066	7111	17,263,832	24.25	0.03
T500_39	6706	16,986,421	6706	17,004,162	22.16	0.10
T500_40	7545	18,633,690	7545	18,641,215	25.98	0.04
Total					**17.08**	**0.21**

Computational results demonstrate the effectiveness and accuracy of DPSO for the new instances that were more complex than the problems practiced by a major body of the literature. The performance of DPSO was consistent for the large-scale problems and was able to produce good feasible solutions in an acceptable timeframe. The obtained results for more complex problems are considered practically reasonable, since, an average deviation of 0.21% from the optima of 500-activity problems is not high. To the best of author's knowledge, the proposed DPSO is one of the first methods capable of obtaining high quality solutions for the large-scale DTCTP within seconds.

A more comprehensive study on the performance of DPSO is given in Section 5.2.4 using a set of benchmark instances acquired from the literature. Based on performance indices demonstrated in Section 5.2.2, effectiveness and efficiency of this method is measured and compared with a new heuristic algorithm which is presented in Section 5.1.3.

CHAPTER 4

PARETO FRONT PARTICLE SWARM OPTIMIZATION METHOD FOR DTCTP

Discussion on the contents of Table 2.1 revealed that the literature is not rich with the more complex Pareto oriented optimization in the domain of TCTP, especially for the discrete version of this problem. This is despite the fact that obtaining non-dominated sets of solutions is widely acknowledged as the ultimate resolution of TCTP analyses. This thesis study emphasizes the importance of discrete and Pareto front TCTPs due to their practical relevance and because they facilitate expressing decision makers preferences so that they can select the best solution based on their own preferences. It was also discovered that there is a lack of Particle Swarm Optimization (PSO) exemplars in the existing literature with the capacity to tackle real-life-size Pareto front DTCT problems. Therefore, followed by an introduction on the Pareto optimality, an efficient and effective multi-objective particle swarm optimization (PSO) model is presented in this chapter. The Pareto front particle swarm optimizer, hereafter called PFPSO, is treated with a simplified heuristic method in order to improve the quality of the initial swarm for an accelerated optimization process. PFPSO is equipped with high capacity to solve large-scale Pareto front problems using unique principles for initialization, representation, and position updating of the particles. The descriptions on background and theoretical properties of PSO and Siemens Approximation Method (SAM) can be found in Sections 3.1 and 3.2, respectively. The mutual aspects of DPSO (Section 3.3) and PFPSO will not be repeated for sake of brevity and only the major modifications

will be pointed out in this chapter. Results of the comparative studies are presented which reveal a performance unmatched by the existing meta-heuristic algorithms.

4.1. Pareto Optimality

Multi-objective problems, such as Pareto front TCTP, often do not have a single optimum solution which makes all the objectives optimal simultaneously. However, for such problems there exist a series of non-dominated solutions. The non-dominated solutions cannot be further improved for one of the objectives and cannot be further worsened for the others (Zhang and Li, 2010). In other words, compared with dominated solutions, non-dominated ones not only are as good as in all measures, but also are better in at least one of them (Zheng et al., 2004). Originally introduced by Vilfredo Pareto, the series consisting of optimal solutions for a multi-objective problem with conflicting objectives is known as the Pareto front or the efficient frontier. The components located on this front are mutually non-dominated with respect to multiple criteria. According to this concept, it is normally not possible to improve one objective without sacrificing at least one other objective.

As such, Pareto front – also known as time-cost curve – extension of the TCT problem is a multi-objective decision making problem, and any of its objectives might reach their optimal amounts at miscellaneous points. Obtaining the Pareto front for TCT problem in essence involves concurrent optimization of budget and deadline TCTPs. Within this domain, the Pareto-dominance optimality is defined according to the following conditions:

- With respect to Eq. (4.1), a decision vector u is said to weakly dominate another solution v if and only if $D_u \leq D_v$ and $C_u \leq C_v$ which is denoted as $u \succeq v$.

$$u \succeq v \ \ iff \ \ \forall y : f_y(u) \leq f_y(v) \tag{4.1}$$

- Regarding Eq. (4.2), solution v is considered to be dominated if and only if there is already another solution u in the efficient frontier such that $D_u \le D_v$ while $C_u \le C_v$, and one of these inequalities holds strictly; this dominance rule is notated as $u \succ v$.

$$u \succ v \;\; iff \;\; \forall y : f_y(u) \le f_y(v) \;\; and \;\; \exists y : f_y(u) < f_y(v) \tag{4.2}$$

- A shown in Eq. (4.3), decision vector u is said to strongly dominate solution v if and only if $D_u < D_v$ and $C_u < C_v$; this dominance condition is denoted as $u \succ\succ v$.

$$u \succ\succ v \;\; iff \;\; \forall y : f_y(u) < f_y(v) \tag{4.3}$$

where; f_y represents the y th objective function. Subsequently, solution u is said to be Pareto optimal as $\forall u, v \in O \;\; if \;\; \nexists v : v \succ u$; union of all of which forms the Pareto front denoted by O.

4.2. Simplified Heuristic

Earlier, Aminbakhsh (2013) presented a method that was similar to SAM which crashed critical activities considering their cost-slopes. Since it is perfectly feasible for the construction schedules to have more than a single critical path, in the course of this method, critical activities on all the multiple parallel critical paths were discovered and the one with the minimum amount of cost-slope was crashed first. However, particularly for large-scale problems with complex networks, the process of identifying the activities having the least cost-slope from a massive pool of critical activities contributes to a substantial computational burden. On the other hand, any variation in selection of the alternatives modifies the project schedule which requires rescheduling of the project for potential changes using the CPM

procedure. That is, this SAM-based method necessitates completing the topological sorting and calculation of CPM iteratively for each cycle of crashing.

In this study, inspired by SAM, a simplified heuristic is developed and embedded into a multi-objective particle swarm optimization method which significantly improves the quality of the initial population. This heuristic is similar to the modified-SAM method introduced in Section 3.2. Though, in the simplified heuristic, instead of determining the complete set of least-cost-slope critical activities for all of the multiple parallel critical paths, one critical activity is selected by examining a single critical path at a time. Crashing is performed by selecting the least-cost-slope activity on the first critical path identified. By doing so, an adequate number of high quality solutions is achieved in a short amount of computation time. The cost-slopes are calculated according to Eq. (3.1) given in Section 3.2. In case there exist multiple least-cost-slope critical activities, this method selects the one leading to a shorter project duration; if the tie is not broken, the activity with the smaller activity number is selected. The procedure is repeated until all of the critical activities in the latest schedule are fully crashed. In order to increase the number of deterministic solutions of high quality, this method reiterates in a similar fashion, by selecting the activity leading to a longer project duration in case of multiple least-cost-slope critical activities. The non-dominated solutions obtained by means of the simplified heuristic are then fed into the particle swarm optimizer.

4.2.1. Case Example

The same case example given in Figure 3.1 of Section 3.3.1 is used to describe the simplified heuristic process. The indirect cost for the case example is assumed to be $1,000/day. For the all-normal (uncrashed) schedule, the project duration is 59 days and the total cost is $206,000 as shown in Table 4.1.

Table 4.1 – Candidate solutions found by simplified heuristic.

# of Schedule	Activity	Crash	Dur. (day)	Direct Cost ($)	Indirect Cost ($)	Total Cost ($)
1		Normal	59	147,000	59,000	206,000
2	1	M3 to M2	47	150,000	47,000	197,000
3	3	M3 to M2	39	162,000	39,000	201,000
4	3	M2 to M1	35	169,000	35,000	204,000
5	1	M2 to M1	33	173,000	33,000	206,600
6	5	M3 to M2	33	175,000	33,000	208,600
7	6	M2 to M1	31	187,000	31,000	218,600

In the Schedule-1 which consists of normal modes, activities 1, 3, and 6 are on the critical path. Among the critical activities, Activity-1 with a cost-slope of $250/day has the minimum cost-slope as shown in Table 4.2, and is crashed first, to its second mode (M2).

Table 4.2 – Cost-slopes of crash modes.

Activity	Crash Mode	Cost-slope ($/Day)
1	M3 to M2	250
1	M2 to M1	2,300
2	M3 to M2	3,000
2	M2 to M1	4,000
3	M3 to M2	1,500
3	M2 to M1	1,750
4	M2 to M1	6,000
5	M3 to M2	2,000
5	M2 to M1	3,200
6	M2 to M1	6,000

The resulting schedule Schedule-2, has a duration of 47 days, and a total cost of $197,000 (Table 4.1). In Schedule-2, activities 1, 3, and 6 are still on the critical path, and among the critical activities Activity-3 has the minimum cost-slope and is crashed to its second mode. In Schedule-3, Activity-3 is crashed again as it has the minimum cost-slope among critical activities. The resulting schedule Schedule-4, has a duration of 35 days, and a total cost of $204,000 and the critical paths include activities 1, 2, 3, 5, and 6, with activities 2, 5, and 6 yet to be crashed.

Similarly, crashing is continued with Activity-5 by crashing it to its second mode. Finally, Activity-6 with the least-cost-slope is crashed to its first mode to obtain a solution of 31 days and total cost of $218,600 as shown in Table 4.1.

The solution that is included in Schedule-1 is dominated by schedules 2, 3, and 4. Likewise, the solution obtained in Schedule-6 is dominated by Schedule-5. The remaining non-dominated solutions that are included in schedules 2, 3, 4, 5, and 7 are recorded as the Pareto front solutions at the end of the first heuristic run. The procedure is repeated while considering the activity leading to a longer project duration in case of a tie between the critical activities during crashing. The second heuristic run also provides exactly the same solutions since in none of the crashing cycles of the heuristic runs a condition for discrimination arises. Although it is not encountered in this specific case problem, should there be a critical path consisting of only Activity-4 and Activity-6, since they both have equal rates of cost-slopes (6,000) for crashing to their first modes, in the first heuristic run the 4^{th} activity, and through the second heuristic run the 6^{th} activity would have been selected. At the end of second heuristic run, the non-dominated solutions are stored, certain percentage of which is transferred to the particle swarm optimizer.

4.3. Pareto front Particle Swarm Optimizer (PFPSO)

Particle swarm optimization is a stochastic, population-based computational optimization method which imitates choreography of bird flocks that forage and fly in unison (for a more comprehensive definition, readers are referred to Section 3.1). Few researches have presented particle swarm optimization methods for single objective DTCTP (Elbeltagi et al., 2005), multi-objective time-cost optimization (Yang, 2007b; Zhang and Li, 2010), times-cost-quality optimization (Fallah-Mehdipour et al., 2012; Zhang and Xing, 2010), and multi-objective time-cost-resource optimization (Ashuri and Tavakolan, 2012). A relatively scant work (Zhang and Li, 2010) is carried out to adopt PSO in locating Pareto front for time-

cost trade-off problems. Existing particle swarm optimization methods are designed to operate in continuous space and are often tested using the small 18-activity problem of Feng et al. (1997). The original single-objective PSO is designed to administer a dominance-based approach for selection of the best particles. This approach cannot guarantee the diversity of the solutions located along the Pareto front, especially for problems with continuous or dense solution spaces. On the other hand, the few studies that extend PSO for optimization of multi-objective problems use aggregating methodology such as weighting or ε-constraint techniques which transform the problem into a single-objective optimization problem with different combinations of weighting and constraints. The method proposed in this chapter aims to achieve diverse non-dominated solutions for Pareto front DTCTP by introducing a new scheme for selection of the best particle by using a Pareto-oriented methodology.

Similar to the DPSO which is covered in Section 3.3, the proposed PFPSO operates in discrete space to present an adequate solution architecture for the DTCTP. While previous research incorporates random generation of the initial particles, the proposed PFPSO incorporates a semi-random initialization scheme presented in Section 3.3. In contrast to DPSO, PFPSO is designed to employ the simplified heuristic introduced in Section 4.2 for generation of the deterministic portion of the initial population. Solutions of simplified heuristic are fed into particle swarm optimizer to improve swarm optimization, by starting the search from high quality solutions. Just like DPSO, the remaining part of the initial swarm population are created randomly to achieve diversification. The maximum percentage of the heuristic solutions that will be included in the initial swarm population is determined by the parameter, percentage (pct). An external archive, O, is dedicated to the PFPSO, to store all of the non-dominated solutions identified. Primarily, this repository stores the non-dominated solutions obtained in the heuristic phase, which is then used to record the new non-dominated particles identified through the subsequent stages of the PFPSO. The remaining initial

population is then generated using a random generation scheme. A controller is implemented in the optimizer to identify the non-dominated particles to be stored in the external archive. As given in Eq. (4.4), for any decision vector u, this controller applies the following criteria based on the measured values of D_u and C_u:

$$\begin{cases} Accept & if & D_u \neq D_v \\ & or & \begin{cases} D_u = D_v \\ C_u \leq C_v \end{cases} \\ Reject & & otherwise \end{cases} \tag{4.4}$$

where; D_v and C_v represent duration and cost of solution v – an existing solution in the archive O. Accordingly, through each iteration the size of the external archive changes dynamically by storing the non-dominated particles while discarding the dominated ones.

Logical sequencing of activities, selection of best positions, objective function evaluations, comparisons, velocity updating, and position updating of the particles are carried out according to Eqs. (3.2), (3.3), (3.4), (3.5), (3.6), (3.7), (3.8), (3.9), (3.10), and (3.11), respectively.

Since the optimal solution for a multi-objective problem comprise a set of non-dominated solutions rather than a single optimum solution, identification of the global best particle from the archived non-dominated ones is crucial to the Pareto-oriented PSO. To the respect of this, PFPSO incorporates a multiple global best approach to determine the P_g position of the particles, in which *gbest* is selected randomly from the external repository of non-dominated particles at each iteration. Hence, in multiple global best approach, equal chance is given to each of the archived solutions in *gbest* selection. Since the objective of Pareto front DTCTP

is to locate the non-dominated time-cost profile over the set of feasible project durations, all the archived non-dominated Pareto front solutions are considered to be of equal quality and are given equal chance to be the *gbest*. Besides, unlike some of the previous multi-objective PSO's that record only parts of the non-dominated solutions for faster calculations (Coello, Pulido, and Lechuga, 2004), PFPSO considers all the non-dominated particles within the archive for selection of the global best particle. The multiple global best approach also enables dynamic exploitation of the archived solutions to locate additional Pareto front solutions and prevents converging to a local optimum.

A 3D illustration of PFPSO's position update scheme for a hypothetical swarm of eight particles for a 3-activity problem is shown in Figure 4.1 and Figure 4.2, in which six non-dominated solutions (P_g's) are archived in the external repository. The first member of the swarm (i_1) flies toward a new position (i_1') using a velocity vector which involves a randomly selected archive member (P_{g4}) and the first particle's *pbest* (P_1) position, as shown in Figure 4.1.

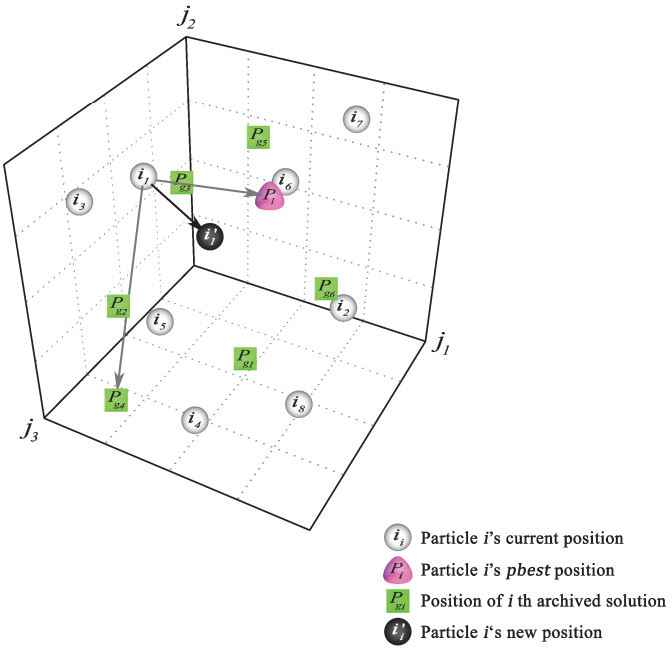


Figure 4.1 – Schematic diagram of PFPSO at time-step t .

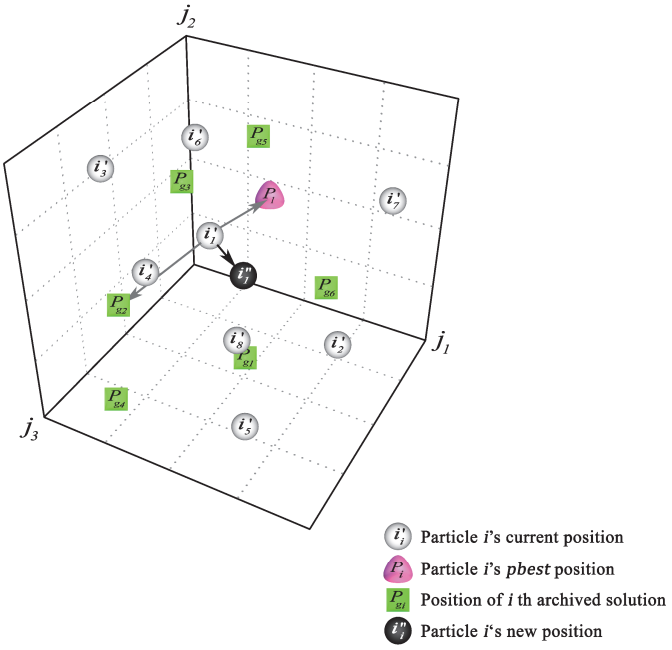


Figure 4.2 – Schematic diagram of PFPSO at time-step t +1.

The external archive is updated according to the new position of the first particle. The positions of the particles i_2 through i_8 and the external archive are updated similarly. The procedure is repeated for the new position of the first particle (i_1') in the next iteration along with the new randomly selected archive member (P_{g2}). The ith particle proceeds to its new position (i_1'') at time-step $t+1$ as shown in Figure 4.2. The optimization process is repeated until the pre-specified number of iterations is reached. PFPSO terminates by returning the ultimate archived non-dominated solutions after discarding the dominated ones according to Eq. (4.5).

$$\forall u, v \in O \;\; if \;\; \exists\, u \succeq v : \; Remove \;\; v \tag{4.5}$$

The simplified heuristic, initial swarm population creation, and position updating stages of the proposed PFPSO are graphically explained as a flowchart in Figure 4.3 and the pseudo-code of this method is demonstrated in Figure 4.4. The proposed PFPSO starts the search with the initial swarm consisting of the deterministic solutions of the simplified heuristic and random particles of PSO, as shown in Figure 4.3. Afterwards, PFPSO iteratively improves the entire population and identifies the non-dominated ones to be stored in the set containing the Pareto solutions.

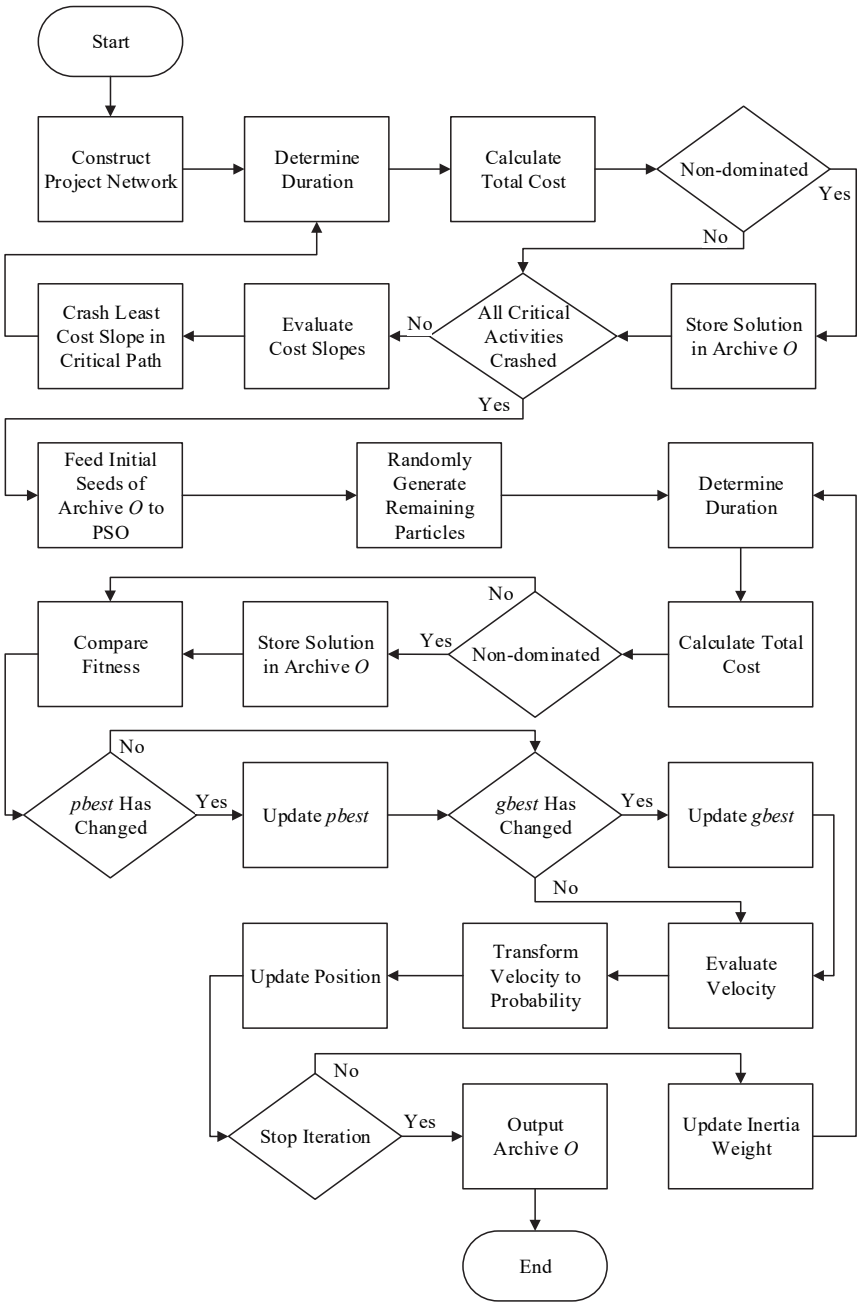


Figure 4.3 – Flowchart of the proposed Pareto front PSO algorithm.

```
Begin;
    For ∀ j ∈ [1, S]
        For ∀ k ∈ [1, m]
            Retrieve Values;
        End;
    End;
    Construct network;
    While ∃ j ∈ Critical-Acts: Fully-Crashed ≠ True ∧ i ≤ pct.N
        Calculate CPM;
        Calculate Dur/Cost;
        If ∀ Sol ∈ O: (Sol.Dur ≠ Dur) ∨ (Sol.Dur = Dur ∧ Sol.Cost ≤ Cost)
            Add Sol to O;
        End;
        Calculate CS;
        Crash Min-CS;
        Set x_i as pbest and gbest;
        Set random Velocity;
    Break;
    For ∀ i ∈ (pct.N, N]
        For ∀ j ∈ Acts
            Set random Position;
            Set random Velocity;
            Set x_i as pbest and gbest;
        End;
        Calculate CPM;
        Calculate Dur/Cost;
        If ∀ Sol ∈ O: (Sol.Dur ≠ Dur) ∨ (Sol.Dur = Dur ∧ Sol.Cost ≤ Cost)
            Add Sol to O;
        End;
    End;
    While t ≤ t_max
        For ∀ i ∈ [1, N]
            Calculate CPM;
            Calculate Dur/Cost;
            For ∀ j ∈ Acts
                Set w;
                If ∀ Sol ∈ O: (Sol.Dur ≠ Dur) ∨ (Sol.Dur = Dur ∧ Sol.Cost ≤ Cost)
                    Add Sol to O;
                End;
                If x_i ≻ P_i
                    Set x_i as pbest;
                End;
                Set a random Sol ∈ O as gbest;
                Calculate Velocity;
```

Figure 4.4 – Pseudo-code of the proposed Pareto front PSO algorithm.

```
            Transform Velocity to Probability;
            Update Position;
        End;
            Transform Velocity to Probability;
            Update Position;
        End;
        Update value of w;
    End;
  Break;
  For ∀ Sol ∈ O: Non-dominated ≠ True
    Remove Sol;
  End;
  Return O;
End;
```

Figure 4.4 – Pseudo-code of the proposed Pareto front PSO algorithm (*Continued*).

4.4. Computational Experiments of PFPSO

Computational experiments are conducted to evaluate the performance of the proposed PFPSO model for solution of Pareto front DTCTP using a set of benchmark instances acquired from the literature. The proposed optimization algorithm is coded in C++ and compiled within Visual Studio 2013 on a 64-bit platform. All of the tests are carried out on a desktop computer with a P9X79 Chipset motherboard, 16 GB 667 MHz DDR3 random-access memory (RAM), Intel Core i7-3.40 GHz CPU, and 64-bit Windows 10 operating system. PFPSO is executed solely (no other software is ran simultaneously) on a single processor and overclocking is not performed.

4.4.1. Parameter Configuration of PFPSO

Evolutionary algorithms are broadly recognized to be very sensitive to configuration of their parameters and the proposed PFPSO is not an exception to this. Therefore, pilot experiments were conducted to determine an adequate set of parameter values for the PFPSO. The parameters of PFPSO were configured based on the experiments which included all combinations of two parameter levels (low

and high) for seven parameters. The pilot experiments revealed that the set of tuned parameters that are summarized in Table 4.3 provided an adequate combination for the PFPSO.

Table 4.3 – Parameter configuration of the PFPSO.

Parameter	Description	Factor Levels		Selected Value
		Low	High	
i	# of Birds	S	$3S$	$3S$
pct	% of deterministic swarm	0.5	0.8	0.8
c_1	Cognitive Parameter	1	2	2
c_2	Social Parameter	1	2	2
$w_{\max}$	Max. Inertia Weight	1	1.2	1
$w_{\min}$	Min. Inertia Weight	0	0.4	0.4
$v_{\max}$	Max. Velocity	3	6	3

250,000 schedules (objective function evaluations) is used as the termination criterion in all of the experiments (Kolisch and Hartmann, 2006; Sonmez and Bettemir, 2012). The performance of the PFPSO is explored for the Pareto front optimization of small, medium, and large-scale benchmark DTCTP problems.

4.4.2. Small-Scale Benchmark Problems

The performance of the proposed Pareto front particle swarm optimizer is first tested with the well-known DTCTP benchmark problems that include 18 activities and up to five time-cost modes. The activity on node (AoN) representation of this instance can be obtained from Feng et al. (1997) and the time-cost data can be attained from Hegazy (1999). As mentioned earlier, even though it has not been pointed out by any other preceding study, it is worth mentioning that this problem is flawed since the cost of third time-cost alternative of the eighth activity must have been selected from the interval $DU(208,215)$; however, the assigned cost is \$200. Nevertheless, in order to conduct a fair comparison with the previous studies, the benchmark problem is used without applying any corrections. The first test problem

(18d) consisted of the DTCTP presented in Feng et al. (1997) which had an indirect cost of $0/day. Problem 18d includes one activity with a single mode, ten activities having three modes, two activities with four modes, and five activities with five modes; accounting for a total of 2.95×10^{9} possible schedules. The small-scale benchmark problems 18e, 18f, and 18g were slightly modified version of the same problem (Hegazy, 1999) with an indirect cost of $0/day, $200/day, and $1,500/day, respectively. 18e, 18f, and 18g all include one activity having two modes, ten activities with three modes, two activities with four modes, and five activities having five modes; accounting for a total of 5.9×10^{9} possible schedules. Although construction projects would typically include an indirect cost, many of the previous studies on Pareto front optimization (Feng et al., 1997; Kandil and El-Rayes, 2006; Afshar et al., 2009; Geem, 2010; Zhang and Li, 2010) used problems with an indirect cost of $0/day in their performance evaluations. Therefore, the same problems were also practiced in the performance evaluation of PFPSO. The optimal solutions for the practiced small-scale problems are determined by applying the mixed-integer programming formulation given in Eqs. (1.1)-(1.6) using AIMMS 4.2 optimization software. The optimal results are also verified by means of a mixed-integer linear programming technique that employs Gurobi solver 6.0.5. This method is explained in detail in Section 5.2.3. Snapshots of the performance of the PFPSO for the four small-scale DTCTP benchmark problems against those of other previous multi-objective optimization techniques is given in Table 4.4. Unavailable values are tabulated as ‘na’ in Table 4.4.

Table 4.4 – Performance comparison of PFPSO for small-scale problems.

Method	Problem	CPU Time (s)	# of Pareto Front Solutions
MAWA-GA (Zheng et al., 2005)	18g	na	4
ACS-TCO (Ng and Zhang, 2008)	18g	na	4
NA-ACO (Afshar et al., 2009)	18e	na	44
	18f	na	18
	18g	na	4
Fuzzy-MOGA (Eshtehardian et al., 2009)	18f	900	18
Harmony Search (Geem, 2010)	18d	5	19
CSMO-PSO (Zhang and Li, 2010)	18e	205	42
ACS (Zhang and Ng, 2012)	18g	na	4
PFPSO (Section 4.3)	18d	2	39
	18e	2	44
	18f	2	18
	18g	2	4

In problem 18d, PFPSO determined 39 non-dominated solutions in 2 seconds which were significantly more than the 19 non-dominated solutions of the harmony search method of Geem (2010). The solutions of the PFPSO for the 19 project durations reported in Geem (2010) are presented along with the solutions of the harmony search method in Table 4.5.

Table 4.5 – Comparison of 19 non-dominated solutions for problem 18d.

Duration (day)	Cost ($)	
	Geem (2010)	PFPSO (Section 4.3)
105	127,320	127,270
106	127,100	127,020
107	126,900	126,770
108	119,415	119,270
109	119,070	119,020
110	118,915	118,770
112	118,620	118,470
114	105,270	105,270
115	105,100	105,020

Table 4.5 – Comparison of 19 non-dominated solutions for problem 18d (*Continued*).

Duration (day)	Cost ($)	
	Geem (2010)	PFPSO (Section 4.3)
116	104,770	104,770
118	104,470	104,470
119	104,270	104,220
120	104,020	103,970
122	103,850	103,720
124	103,070	103,070
125	102,908	102,820
126	102,708	102,570
128	102,400	102,320
131	102,320	102,170

The results of Table 4.5 can be summarized as follows. The solutions of the PFPSO in comparison with those of the harmony search are of higher quality. This comparison shows PFPSO is able to achieve better solutions for 15 project durations while obtaining the same results for the remaining four durations.

The combined scheme-based multi-objective particle swarm optimization (CSMO-PSO) method of Zhang and Li (2010) was able to obtain 42 non-dominated solutions for 18e in 205 seconds. PFPSO located 44 non-dominated solutions in 2 seconds for the same problem. The proposed Pareto front particle swarm optimizer captured larger number of non-dominated solutions in a significantly less computation time compared with CSMO-PSO of Zhang and Li (2010). The multi-colony ant algorithm of Afshar et al. (2009) also achieved 44 non-dominated solutions for problem 18e, however, the computational time requirement of this method was not reported.

PFPSO was able to capture 18 non-dominated solutions for 18f in 2 seconds. Likewise, the multi-objective genetic algorithm (MOGA) of Eshtehardian et al. (2009) is shown to be able to achieve 18 non-dominated solutions for the same problem, but it requires 900 seconds to locate the Pareto front. One genetic (Zheng

et al., 2005) and three ant colony algorithms (Ng and Zhang, 2008; Afshar et al., 2009; Zhang and Ng, 2012) reported four non-dominated solutions for 18g. Computation times of the methods were not included. PFPSO also obtained four non-dominated solutions for 18g in 2 seconds. All of the four non-dominated solutions of the PFPSO were better than those of Zheng et al. (2005), and one of the non-dominated solutions was better than the solution of Ng and Zhang (2008) and Zhang and Ng (2012) as shown in Table 4.6.

Table 4.6 – Comparison of four non-dominated solutions for problem 18g.

Duration (day)	Zheng et al. (2005)	Ng and Zhang (2008)	Zhang and Ng (2012)	PFPSO (Section 4.3)
100	287,720	283,320	283,320	283,320
101	284,020	279,820	279,820	279,820
104	280,020	276,320	276,320	276,320
110	273,720	271,320	271,320	271,270

The computational experiments reveal that the performance of the PFPSO on the well-known small-scale benchmark problems is unmatched by the previous meta-heuristic methods. Results indicate that PFPSO can locate large number of high quality Pareto front solutions. Although the test configuration details, such as the CPU speed, are not available for majority of the previous approaches, the results reveal that the computation time requirement of PFPSO is significantly less than the time requirement of the existing methods. The computational efficiency of PFPSO is more prominent in comparison with the existing methods that are capable of locating large number of Pareto front solutions for the small-scale problems.

In addition to the comparative study with the existing multi-objective methods, the convergence degree of PFPSO is also investigated by means of the optimal Pareto front solutions obtained using mixed-integer programming technique. The optimal solutions prove that the time-cost curves captured by PFPSO for problems 18d, 18e, 18f, and 18g were indeed true Pareto fronts consisting of the optimum non-

dominated solutions. True Pareto fronts generated by PFPSO for problems 18d, 18e, 18f, and 18g are illustrated in Table 4.7, Table 4.8, Table 4.9, and Table 4.10, respectively.

Table 4.7 - Performance of PFPSO for problem 18d.

Pareto Front				Mode Selection																	
Duration	Cost ($)		Deviation	Activities																	
(day)	Optimal	PFPSO	(%)	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*
104	132,270	132,270	0	5	1	1	1	2	3	1	1	5	3	1	4	1	1	1	1	3	3
105	127,270	127,270	0	5	1	1	1	1	3	1	1	5	3	1	4	1	1	1	1	3	3
106	127,020	127,020	0	4	1	1	1	1	3	1	1	5	3	1	4	1	1	1	1	3	3
107	126,770	126,770	0	3	1	1	1	1	3	1	1	5	3	1	4	1	1	1	1	3	3
108	119,270	119,270	0	5	1	1	1	1	2	1	1	5	3	1	4	1	1	1	1	3	3
109	119,020	119,020	0	4	1	1	1	1	2	1	1	5	3	1	4	1	1	1	1	3	3
110	118,770	118,770	0	3	1	1	1	1	2	1	1	5	3	1	4	1	1	1	1	3	3
112	118,470	118,470	0	5	1	1	1	1	2	1	1	5	3	1	4	1	1	1	1	2	3
113	118,220	118,220	0	4	1	1	1	1	2	1	1	5	3	1	4	1	1	1	1	2	3
114	105,270	105,270	0	5	1	1	1	1	1	1	1	5	3	1	4	1	1	1	1	3	3
115	105,020	105,020	0	4	1	1	1	1	1	1	1	5	3	1	4	1	1	1	1	3	3
116	104,770	104,770	0	3	1	1	1	1	1	1	1	5	3	1	4	1	1	1	1	3	3
118	104,470	104,470	0	5	1	1	1	1	1	1	1	5	3	1	4	1	1	1	1	2	3
119	104,220	104,220	0	4	1	1	1	1	1	1	1	5	3	1	4	1	1	1	1	2	3
120	103,970	103,970	0	3	1	1	1	1	1	1	1	5	3	1	4	1	1	1	1	2	3
122	103,720	103,720	0	3	1	1	1	1	1	1	1	5	3	1	3	1	1	1	1	2	3
124	103,070	103,070	0	5	1	1	1	1	1	1	1	5	3	1	4	1	1	1	1	1	3
125	102,820	102,820	0	4	1	1	1	1	1	1	1	5	3	1	4	1	1	1	1	1	3
126	102,570	102,570	0	3	1	1	1	1	1	1	1	5	3	1	4	1	1	1	1	1	3
128	102,320	102,320	0	3	1	1	1	1	1	1	1	5	3	1	3	1	1	1	1	1	3
131	102,170	102,170	0	2	1	1	1	1	1	1	1	5	3	1	4	1	1	1	1	1	3
132	101,970	101,970	0	3	1	1	1	1	1	1	1	5	3	1	4	1	1	1	1	1	2
133	101,820	101,820	0	4	1	1	1	1	1	1	1	5	3	1	1	1	1	1	1	1	3
134	101,570	101,570	0	3	1	1	1	1	1	1	1	5	3	1	1	1	1	1	1	1	3
137	101,510	101,510	0	3	1	1	1	1	1	1	1	4	3	1	1	1	1	1	1	1	3
138	101,470	101,470	0	5	1	1	1	1	1	1	1	5	3	1	1	1	1	1	1	1	2
139	101,170	101,170	0	2	1	1	1	1	1	1	1	5	3	1	1	1	1	1	1	1	3
140	100,970	100,970	0	3	1	1	1	1	1	1	1	5	3	1	1	1	1	1	1	1	2
142	100,870	100,870	0	1	1	1	1	1	1	1	1	5	3	1	1	1	1	1	1	1	3
143	100,770	100,770	0	3	1	1	1	1	1	1	1	5	3	1	1	1	1	1	1	1	1

Table 4.7 - Performance of PFPSO for problem 18d (*Continued*).

Pareto Front				Mode Selection																	
Duration	**Cost ($)**		**Deviation**	**Activities**																	
(day)	**Optimal**	**PFPSO**	**(%)**	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***	***13***	***14***	***15***	***16***	***17***	***18***
145	100,570	100,570	0	2	1	1	1	1	1	1	1	5	3	1	1	1	1	1	1	1	2
148	100,270	100,270	0	1	1	1	1	1	1	1	1	5	3	1	1	1	1	1	1	1	2
151	100,070	100,070	0	1	1	1	1	1	1	1	1	5	3	1	1	1	1	1	1	1	1
154	100,010	100,010	0	1	1	1	1	1	1	1	1	4	3	1	1	1	1	1	1	1	1
156	99,950	99,950	0	1	1	1	1	1	1	1	1	3	3	1	1	1	1	1	1	1	1
158	99,900	99,900	0	1	1	1	1	1	1	1	1	3	2	1	1	1	1	1	1	1	1
159	99,870	99,870	0	1	1	1	1	1	1	1	1	2	2	1	1	1	1	1	1	1	1
161	99,820	99,820	0	1	1	1	1	1	1	1	1	1	2	1	1	1	1	1	1	1	1
169	99,740	99,740	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
APD (%)			**0**																		

Table 4.8 – Performance of PFPSO for problem 18e.

Pareto Front				Mode Selection																	
Duration	**Cost ($)**		**Deviation**	**Activities**																	
(day)	**Optimal**	**PFPSO**	**(%)**	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***	***13***	***14***	***15***	***16***	***17***	***18***
100	133,320	133,320	0	5	1	1	1	2	3	1	1	5	3	2	4	1	1	2	1	3	3
101	128,320	128,320	0	5	1	1	1	1	3	1	1	5	3	2	4	1	1	2	1	3	3
102	128,070	128,070	0	4	1	1	1	1	3	1	1	5	3	2	4	1	1	2	1	3	3
103	127,820	127,820	0	3	1	1	1	1	3	1	1	5	3	2	4	1	1	2	1	3	3
104	120,320	120,320	0	5	1	1	1	1	2	1	1	5	3	2	4	1	1	2	1	3	3
105	120,070	120,070	0	4	1	1	1	1	2	1	1	5	3	2	4	1	1	2	1	3	3
106	119,820	119,820	0	3	1	1	1	1	2	1	1	5	3	2	4	1	1	2	1	3	3
107	119,770	119,770	0	3	1	1	1	1	2	1	1	5	3	1	4	1	1	2	1	3	3
108	119,270	119,270	0	5	1	1	1	1	2	1	1	5	3	1	4	1	1	1	1	3	3
109	119,020	119,020	0	4	1	1	1	1	2	1	1	5	3	1	4	1	1	1	1	3	3
110	106,270	106,270	0	5	1	1	1	1	1	1	1	5	3	1	4	1	1	2	1	3	3
111	106,020	106,020	0	4	1	1	1	1	1	1	1	5	3	1	4	1	1	2	1	3	3
112	105,770	105,770	0	3	1	1	1	1	1	1	1	5	3	1	4	1	1	2	1	3	3
114	105,270	105,270	0	5	1	1	1	1	1	1	1	5	3	1	4	1	1	1	1	3	3
115	105,020	105,020	0	4	1	1	1	1	1	1	1	5	3	1	4	1	1	1	1	3	3
116	104,770	104,770	0	3	1	1	1	1	1	1	1	5	3	1	4	1	1	1	1	3	3
118	104,470	104,470	0	5	1	1	1	1	1	1	1	5	3	1	4	1	1	1	1	2	3
119	104,220	104,220	0	4	1	1	1	1	1	1	1	5	3	1	4	1	1	1	1	2	3
120	103,970	103,970	0	3	1	1	1	1	1	1	1	5	3	1	4	1	1	1	1	2	3

Table 4.8 – Performance of PFPSO for problem 18e (*Continued*).

Pareto Front				Mode Selection																	
Duration	**Cost ($)**		**Deviation**	**Activities**																	
(day)	**Optimal**	**PFPSO**	**(%)**	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***	***13***	***14***	***15***	***16***	***17***	***18***
121	103,820	103,820	0	4	1	1	1	1	1	1	1	5	3	1	4	1	1	2	1	1	3
122	103,570	103,570	0	3	1	1	1	1	1	1	1	5	3	1	4	1	1	2	1	1	3
124	103,070	103,070	0	5	1	1	1	1	1	1	1	5	3	1	4	1	1	1	1	1	3
125	102,820	102,820	0	4	1	1	1	1	1	1	1	5	3	1	4	1	1	1	1	1	3
126	102,570	102,570	0	3	1	1	1	1	1	1	1	5	3	1	4	1	1	1	1	1	3
128	102,320	102,320	0	3	1	1	1	1	1	1	1	5	3	1	3	1	1	1	1	1	3
131	102,170	102,170	0	2	1	1	1	1	1	1	1	5	3	1	4	1	1	1	1	1	3
132	101,970	101,970	0	3	1	1	1	1	1	1	1	5	3	1	4	1	1	1	1	1	2
133	101,820	101,820	0	4	1	1	1	1	1	1	1	5	3	1	1	1	1	1	1	1	3
134	101,570	101,570	0	3	1	1	1	1	1	1	1	5	3	1	1	1	1	1	1	1	3
137	101,510	101,510	0	3	1	1	1	1	1	1	1	4	3	1	1	1	1	1	1	1	3
138	101,470	101,470	0	3	1	1	1	1	1	1	1	5	3	1	2	1	1	1	1	1	2
139	101,170	101,170	0	2	1	1	1	1	1	1	1	5	3	1	1	1	1	1	1	1	3
140	100,970	100,970	0	3	1	1	1	1	1	1	1	5	3	1	1	1	1	1	1	1	2
142	100,870	100,870	0	1	1	1	1	1	1	1	1	5	3	1	1	1	1	1	1	1	3
143	100,770	100,770	0	3	1	1	1	1	1	1	1	5	3	1	1	1	1	1	1	1	1
145	100,570	100,570	0	2	1	1	1	1	1	1	1	5	3	1	1	1	1	1	1	1	2
148	100,270	100,270	0	1	1	1	1	1	1	1	1	5	3	1	1	1	1	1	1	1	2
151	100,070	100,070	0	1	1	1	1	1	1	1	1	5	3	1	1	1	1	1	1	1	1
154	100,010	100,010	0	1	1	1	1	1	1	1	1	4	3	1	1	1	1	1	1	1	1
156	99,950	99,950	0	1	1	1	1	1	1	1	1	3	3	1	1	1	1	1	1	1	1
158	99,900	99,900	0	1	1	1	1	1	1	1	1	3	2	1	1	1	1	1	1	1	1
159	99,870	99,870	0	1	1	1	1	1	1	1	1	2	2	1	1	1	1	1	1	1	1
161	99,820	99,820	0	1	1	1	1	1	1	1	1	1	2	1	1	1	1	1	1	1	1
169	99,740	99,740	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
APD (%)			**0**																		

Table 4.9 – Performance of PFPSO for problem 18f.

Pareto Front				Mode Selection																	
Duration	**Cost ($)**		**Deviation**	**Activities**																	
(day)	**Optimal**	**PFPSO**	**(%)**	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***	***13***	***14***	***15***	***16***	***17***	***18***
100	153,320	153,320	0	5	1	1	1	2	3	1	1	5	3	2	4	1	1	2	1	3	3
101	148,520	148,520	0	5	1	1	1	1	3	1	1	5	3	2	4	1	1	2	1	3	3
102	148,470	148,470	0	4	1	1	1	1	3	1	1	5	3	2	4	1	1	2	1	3	3

Table 4.9 – Performance of PFPSO for problem 18f (*Continued*).

Pareto Front				Mode Selection																
Duration	**Cost ($)**		**Deviation**	**Activities**																
(day)	**Optimal**	**PFPSO**	**(%)**	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***	***13***	***14***	***15***	***16***	***17***	***18***
103	148,420	148,420	0	3	1	1	1	1	3	1	1	5	3	2	4	1	1	2	1	3	3
104	141,120	141,120	0	5	1	1	1	1	2	1	1	5	3	2	4	1	1	2	1	3	3
105	141,070	141,070	0	4	1	1	1	1	2	1	1	5	3	2	4	1	1	2	1	3	3
106	141,020	141,020	0	3	1	1	1	1	2	1	1	5	3	2	4	1	1	2	1	3	3
108	140,870	140,870	0	5	1	1	1	1	2	1	1	5	3	1	4	1	1	1	1	3	3
109	140,820	140,820	0	4	1	1	1	1	2	1	1	5	3	1	4	1	1	1	1	3	3
110	128,270	128,270	0	5	1	1	1	1	1	1	1	5	3	1	4	1	1	2	1	3	3
111	128,220	128,220	0	4	1	1	1	1	1	1	1	5	3	1	4	1	1	2	1	3	3
112	128,170	128,170	0	3	1	1	1	1	1	1	1	5	3	1	4	1	1	2	1	3	3
114	128,070	128,070	0	5	1	1	1	1	1	1	1	5	3	1	4	1	1	1	1	3	3
115	128,020	128,020	0	4	1	1	1	1	1	1	1	5	3	1	4	1	1	1	1	3	3
116	127,970	127,970	0	3	1	1	1	1	1	1	1	5	3	1	4	1	1	1	1	3	3
124	127,870	127,870	0	5	1	1	1	1	1	1	1	5	3	1	4	1	1	1	1	1	3
125	127,820	127,820	0	4	1	1	1	1	1	1	1	5	3	1	4	1	1	1	1	1	3
126	127,770	127,770	0	3	1	1	1	1	1	1	1	5	3	1	4	1	1	1	1	1	3
APD (%)			**0**																		

Table 4.10 – Performance of PFPSO for problem 18g.

Pareto Front				Mode Selection																
Duration	**Cost ($)**		**Deviation**	**Activities**																
(day)	**Optimal**	**PFPSO**	**(%)**	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***	***13***	***14***	***15***	***16***	***17***	***18***
100	283,320	283,320	0	5	1	1	1	2	3	1	1	5	3	2	4	1	1	2	1	3	3
101	279,820	279,820	0	5	1	1	1	1	3	1	1	5	3	2	4	1	1	2	1	3	3
104	276,320	276,320	0	5	1	1	1	1	2	1	1	5	3	2	4	1	1	2	1	3	3
110	271,270	271,270	0	5	1	1	1	1	1	1	1	5	3	1	4	1	1	2	1	3	3
APD (%)			**0**																		

4.4.3. Medium-Scale Benchmark Problem

Kandil and El-Rayes (2006) created medium-scale DTCTP problems by using the time-cost alternatives of the 18-activity problem of Feng et al. (1997) and by copying a slightly modified version of the same network ten times in serial. The

network, which is used to create medium-scale and large-scale (Section 4.4.4) problems is demonstrated in Figure 4.5 (Kandil, 2005).

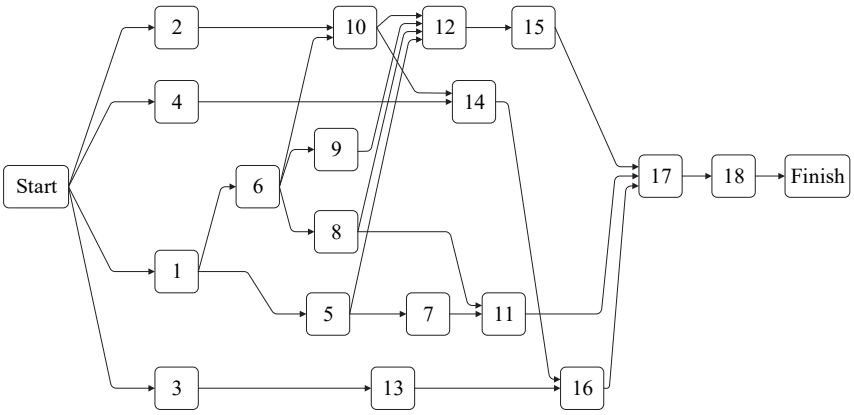


Figure 4.5 - Network diagram of the core problem for medium-scale and large-scale problems.

The medium-scale problem consists of 180 activities which is practiced with an indirect cost rate of $0/day. The performance of the proposed PFPSO for 180-activity problem is compared with the performances of the global parallel and coarse grained genetic algorithm approaches of Kandil and El-Rayes (2006) in terms of number of Pareto solutions and computation time requirements. The global parallel genetic algorithm module was implemented using 50 processors on the tungsten supercomputing cluster, which was composed of 640 Dell PowerEdge 1750 servers, each with two Intel Xeon 3.2 GHz processors, 1.5 MB of cache memory, and a total of 3 GB of SDRAM (Kandil and El-Rayes, 2006). The supercomputing cluster had a peak performance of 6.4 Gflops. As summarized in Table 4.11, the global parallel GA of Kandil and El-Rayes (2006) obtained 267 Pareto solutions for 180-activity problem running on a single processor (Kandil, 2005) in 14,688 seconds.

Table 4.11 – Comparison of the results for 180-activity problem.

Algorithm	# of Processor(s)	CPU Time (s)	# of Pareto Front Solutions
GP-GA (Kandil and El-Rayes, 2006)	1	14,688	267*
GP-GA (Kandil and El-Rayes, 2006)	50	2,556	267*
CG-GA (Kandil and El-Rayes, 2006)	50	684	68*
PFPSO (Section 4.3)	1	21	304

*Kandil (2005)

The computation time requirement of the global parallel GA was reduced to 2,556 seconds by using a cluster of 50 processors. The coarse-grained GA, on the other hand, located 68 Pareto solutions in 684 seconds over a cluster of 50 processors. The proposed particle swarm Pareto front optimizer was able to capture 304 Pareto solutions in 21 seconds on a single processor. PFPSO was able to position a rather larger set of non-dominated solutions along the efficient frontier than all of the three GA-based approaches of Kandil and El-Rayes (2006), in a significantly less computation time.

In addition to the comparative study with the existing parallel GA methods of Kandil and El-Rayes (2006), the accuracy of PFPSO is also investigated by means of the optimal costs of Pareto front solutions obtained using mixed-integer programming technique. The optimal solutions reveal that the proposed PFPSO was able to capture high quality Pareto front solutions with an average deviation of only 0.1% from the optimal costs. Results of PFPSO for the medium-scale problem is summarized in Table 4.12. However, for sake of brevity only the first nine solutions out of a total of 304 showing mode selection of only 17 activities out of 180 are pointed out in this table.

Table 4.12 – Performance of PFPSO for 180-activity problem.

Pareto Front				Mode Selection																	
Duration	**Cost ($)**		**Deviation**	**Activities**																	
(day)	**Optimal**	**PFPSO**	**(%)**	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***	***13***	***14***	***15***	***16***	***17***	**…**
1040	1,335,650	1,339,700	0.303	5	1	1	1	2	3	1	5	5	3	2	4	1	3	1	3	3	
1041	1,330,385	1,334,700	0.324	5	1	1	1	1	3	1	5	5	3	2	4	1	3	1	3	3	
1042	1,325,120	1,329,700	0.346	5	1	1	1	1	3	1	5	5	3	2	4	1	3	1	3	3	
1043	1,319,855	1,324,700	0.367	5	1	1	1	1	3	1	5	5	3	2	4	1	3	1	3	3	
1044	1,314,590	1,319,700	0.389	5	1	1	1	1	3	1	5	5	3	2	4	1	3	1	3	3	
1045	1,309,325	1,314,700	0.411	5	1	1	1	1	3	1	5	5	3	2	4	1	3	1	3	3	
1046	1,304,060	1,309,100	0.386	5	1	1	1	1	3	1	5	5	3	2	4	1	3	1	3	3	
1047	1,298,795	1,304,100	0.408	5	1	1	1	1	3	1	5	5	3	2	4	1	3	1	3	3	
1048	1,293,530	1,299,100	0.431	5	1	1	1	1	3	1	5	5	3	2	4	1	3	1	3	3	
⋮																					
APD (%)			**0.095**																		

4.4.4. Large-Scale Benchmark Problems

Experiments on large-scale problems comprise the 360-activity and 720-activity benchmark problems created by Kandil and El-Rayes (2006). Similar to their medium-scale problem, the time-cost alternatives of the 18-activity problem of Feng et al. (1997) were duplicated for networks of size 360 and 720 which were generated by copying the same network shown in Figure 4.5 several times in serial.

Both of the large-scale problems are practiced with an indirect cost rate of $0/day. The performances of the proposed PFPSO for 360-activity and 720-activity problems are compared with the performances of the global parallel and coarse grained genetic algorithms of Kandil and El-Rayes (2006) with regard to number of Pareto solutions and computational time. The global parallel genetic algorithm module was implemented using a supercomputing cluster which is described in Section 4.4.3. Performance of the results for the large-scale 360-activity instance is presented in Table 4.13.

Table 4.13 – Comparison of the results for 360-activity problem.

Algorithm	# of Processor(s)	CPU Time (s)	# of Pareto Front Solutions
GP-GA (Kandil and El-Rayes, 2006)	1	75,096	232
GP-GA (Kandil and El-Rayes, 2006)	50	10,404	232
CG-GA (Kandil and El-Rayes, 2006)	50	1,836	94
PFPSO (Section 4.3)	1	43	536

Over a cluster of 50 processors, the global parallel GA was able to obtain 232 non-dominated solutions in 10,404 seconds, whereas the coarse-grained GA achieved 94 non-dominated solutions in 1,836 seconds for 360-activity problem. The proposed particle swarm Pareto front optimization method was able to capture 536 non-dominated solutions in 43 seconds for the same problem by running on a single processor. The exceptional performance of PFPSO was consistent for larger problems as it captured larger set of non-dominated solutions than all of the three GA-based approaches of Kandil and El-Rayes (2006), within a considerably less computational effort. Results of PFPSO for the 360-activity problem is summarized in Table 4.14. Though, in the consideration of briefness only the first nine solutions out of a total of 536 showing mode selection of only 17 activities out of 360 are tabulated as follows.

Table 4.14 – Non-dominated solutions of PFPSO for 360-activity problem.

Pareto Front		**Mode Selection**																	
Duration	**Cost**	**Activities**																	
(day)	**($)**	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***	***13***	***14***	***15***	***16***	***17***	**…**
2080	2,679,345	5	1	1	1	2	3	1	5	5	3	2	4	1	3	1	3	3	
2081	2,674,345	5	1	1	1	1	3	1	5	5	3	2	4	1	3	1	3	3	
2082	2,668,145	5	1	1	1	1	3	1	5	5	3	2	4	1	2	1	3	3	
2083	2,663,195	5	1	1	1	1	3	1	5	5	3	2	4	1	2	1	3	3	
2084	2,658,145	5	1	1	1	1	3	1	5	5	3	2	4	1	2	1	3	3	
2085	2,653,145	5	1	1	1	1	3	1	5	5	3	2	4	1	2	1	3	3	
2086	2,648,145	5	1	1	1	1	3	1	5	5	3	2	4	1	2	1	3	3	
2087	2,643,145	5	1	1	1	1	3	1	5	5	3	2	4	1	2	1	3	3	
2088	2,638,195	5	1	1	1	1	3	1	5	5	3	2	4	1	2	1	3	3	
⋮																			

The successful performance of the PFPSO is also consistent for the large-scale 720-activity benchmark problem, as shown in Table 4.15.

Table 4.15 – Comparison of the results for 360-activity problem.

Algorithm	# of Processor(s)	CPU Time (s)	# of Pareto Front Solutions
GP-GA (Kandil and El-Rayes, 2006)	1	491,400	303
GP-GA (Kandil and El-Rayes, 2006)	50	55,296	303
CG-GA (Kandil and El-Rayes, 2006)	50	7,092	132
PFPSO (Section 4.3)	1	92	1022

PFPSO was able to capture 1022 non-dominated solutions in 92 seconds for the 720-activity problem with a single processor on a desktop computer. Whereas, over a cluster of 50 processors, the global parallel GA provided 303 Pareto solutions in 55,296 seconds, and the coarse-grained GA obtained 132 non-dominated solutions in 7,091 seconds for 720-activity problem. Results of PFPSO for the 720-activity problem is summarized in Table 4.16. However, due to length considerations only the first nine solutions out of a total of 1022 showing mode selection of only 17 activities out of 720 are illustrated as follows.

Table 4.16 – Non-dominated solutions of PFPSO for 720-activity problem.

Pareto Front		**Mode Selection**																
Duration	**Cost**	**Activities**																
(day)	**($)**	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***	***13***	***14***	***15***	***16***	***17***	**…**
4160	5,358,800	5	1	1	1	2	3	1	5	5	3	2	4	1	3	1	3	3	
4161	5,353,800	5	1	1	1	1	3	1	5	5	3	2	4	1	3	1	3	3	
4162	5,348,800	5	1	1	1	1	3	1	5	5	3	2	4	1	3	1	3	3	
4163	5,343,650	5	1	1	1	1	3	1	5	5	3	2	4	1	3	1	3	3	
4164	5,338,800	5	1	1	1	1	3	1	5	5	3	2	4	1	3	1	3	3	
4165	5,333,800	5	1	1	1	1	3	1	5	5	3	2	4	1	3	1	3	3	
4166	5,328,750	5	1	1	1	1	3	1	5	5	3	2	4	1	3	1	3	3	
4167	5,323,800	5	1	1	1	1	3	1	5	5	3	2	4	1	3	1	3	3	
4168	5,318,800	5	1	1	1	1	3	1	5	5	3	2	4	1	3	1	3	3	
⋮																			

Obtained by PFPSO, 304, 536, and 1022 non-dominated solutions constituting the Pareto fronts of 180-activity, 360-activity, and 720-activity problems are illustrated in Figure 4.6.

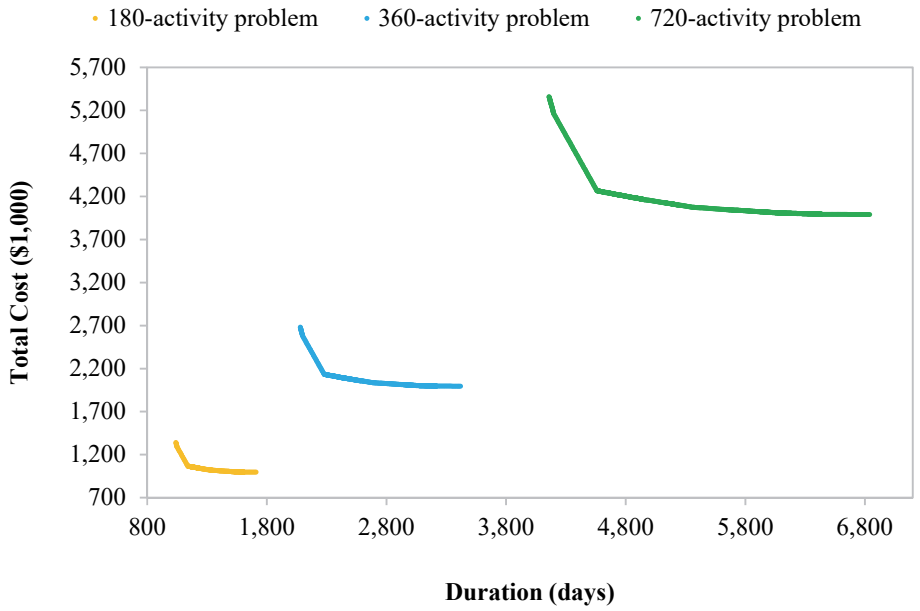


Figure 4.6 – Pareto fronts located by PFPSO for 180, 360, and 720-activity problems.

The mixed integer model mentioned in Section 3.4.2 was also used to determine the optimal costs for both 360-activity and 720-activity problems; nevertheless, optimal costs of these problems were not obtained within 72 hours. As is clear, not only the computation time requirement of the proposed Pareto front particle swarm optimizer was significantly less than earlier approaches but was also able to produce a large number of good feasible Pareto front solutions for the large-scale problems. Due to the effectiveness and efficiency of the proposed particle swarm optimization method, it is expected to contribute to optimal planning of real-life-scale construction projects. To the best of author's knowledge, the proposed PFPSO is one of the first methods capable of capturing high quality Pareto solutions for the large-scale DTCTP within seconds.

A more comprehensive study on the performance of PFPSO is given in Section 5.2.5 using new sets of RanGen2 instances discussed in Section 5.2.1. Based on performance indices demonstrated in Section 5.2.2, effectiveness and efficiency of this approach is measured and compared with a new heuristic algorithm which is presented in Section 5.1.4.

CHAPTER 5

COST-SLOPE HEURISTIC METHOD FOR DTCTP

Respecting the state of the existing research on TCTP in the construction industry, it can be observed that the meta-heuristic approaches are the most prominent methods used in a multitude of studies. Nonetheless, it is broadly acknowledged that evolutionary algorithms are very sensitive to configuration of their parameters. In real-life situations, the experimental process for configuration of parameters for best values may become a tedious and arduous task. Yet, the parameters need to be retuned for each new problem at hand which might reduce the practicability of the meta-heuristic approaches. On the other hand, experimentation of DPSO (Section 3.4) and PFPSO (Section 4.4) revealed that the exceptional performances of the noted approaches were largely resulting from their heuristic modules. It was also discovered that there exist only a handful of heuristic exemplars in the existing literature which is limited to the methods proposed by Fondahl (1961), Siemens (1971), Goyal (1975), Moselhi (1993), and Bettemir and Birgonul (2017). However, none of the earlier heuristics mentioned have the capacity to tackle real-life-scale Pareto front discrete TCT problems since most of them are designed for rather simple and continuous deadline problems. The discrete Pareto front TCTP is considered as the most salient type this trade-off problem due to its practical relevance and also because of its potential for articulation of managers' propensities which provides them with tools for selection of the best solution with respect to their preferences. These are some of the very reasons that inspired development of a new heuristic approach, hereafter called Cost-Slope Heuristic, for solution of TCTP in this thesis which is discussed in Section 5.1.

Moreover, apart from the fact that the existing approaches have seldom been applied for solution of large-scale DTCTPs, it is interpreted that the dearth of real-life-scale problems could possibly be another major reason for the lack of studies on realistic problems. Despite the fact that some studies have included problems including up to 720, 2000, and 6300 activities, all of the employed large-scale problems are generated using small-scale base networks and are generated by copying the core problem in serial several times; hence, these problems are believed to have limitations in reflecting the complexity of the real-life construction projects. To the respect of this, for better evaluation of the capabilities of the proposed optimization models, new sets of multi-mode large-scale DTCT problems have been generated in this thesis by means of random network generator, viz., RanGen2. The systematically generated large-scale instances comprise complex networks and realistic sets of time-cost alternatives which are elucidated in Section 5.2.1.

On the other hand, it is also observed that the majority of the earlier research on DTCTP not only employ instances with small problem networks, but also the detailed performance evaluation on accuracy and efficiency of most of the presented approaches appear to be lacking, especially for the Pareto front problem. Unlike a large body of the existing literature, performance measurements are carried out by taking a more holistic approach that involves a set of performance comparison indices. In order to compare methods rigorously and to measure performances on a quantitative bases, efficiency, accuracy, diversity, and cardinality of the obtained solutions are evaluated using the indices explained in Section 5.2.2.

Meanwhile, it is not possible to accurately assess quality of the solutions obtained from heuristic or meta-heuristic algorithms short of exact procedures since they are the only approaches guaranteeing optimality of the solutions. Owing to this very reason, they play a crucial role in performance evaluation of non-exact optimization algorithms. Therefore, a Mixed-Integer Linear programming technique with unique

features is also proposed for obtaining the optimal results of the practiced instances which is described in Section 5.2.3.

The computational experiments include comparative studies on the performance of the Cost-Slope Heuristic (Section 5.1), DPSO (Section 3.3) and PFPSO (Section 4.3). Benchmark instances attained from the literature as well the new sets of RanGen2 instances presented in Section 5.2.1 are used to conduct the comparative studies which prove remarkable efficiency and exceptional efficacy of the proposed Cost-Slope Heuristic for real-life practices.

5.1. Cost-Slope Heuristic

A heuristic method is defined as a logical sequence of steps – relying on rules of thumb, execution of which yields an optimum or near-optimum solution (Demeulemeester and Herroelen, 2002). The literature on heuristic algorithms for the TCTP is virtually restricted to Fondahl (1961), Siemens (1971), Goyal (1975), and Moselhi (1993). These methods assume linear time-cost relationships and are incapable of handling discrete TCT problems. The most recent heuristic approach is proposed by Bettemir and Birgonul (2017) which is one of the very few methods designed for unraveling the discrete TCT problems. Their method which is called Network Analysis Algorithm (NAA) can tackle deadline and cost minimization DTCT problems. To the best of author's knowledge, there exists no alternative multi-objective heuristic method designed for Pareto oriented optimization of DTCTP other than the SAM-based approaches proposed by the author previously. Two concepts are introduced herein for reducing the size of the network and for faster CPM calculations which are described in detail in Section 5.1.1.1 and Section 5.1.1.2, respectively. The details of the methodology implemented to develop the proposed Cost-Slope Heuristic, abridged as CS-Heuristic, for deadline and Pareto front TCTP are explained in Section 5.1.3 and Section 5.1.4, respectively.

5.1.1. Network Reduction Techniques

Just as an increase in the scale of the problem causes an exponential growth in computational burden, a decrease in scale results in an exponential decline in computational burden. The simplification of the problem which is known as the network reduction technique, despite maintaining the abovementioned utility, has not received sufficient attention. Based on the methods proposed by Rothfarb, Frank, Rosebaum, Steiglitz, and Kleitman (1970) and Frank, Frisch, Van Slyke, and Chou (1971) two clever network reduction techniques are proposed in this chapter which will be explained in the ensuing sections. Unlike preceding work on this area which considered AoA networks, this thesis study proposes simplification methods for AoN notation systems. According to the proposed method, serial/parallel reducible problems can be simplified through replacing certain activities by merging them into an equivalent activity. In order to elucidate these concepts, a numerical example is presented based on the sample case problem introduced in Figure 3.1 of Section 3.3.1. This instance is exercised by assuming a completion deadline of 45 days, with delay penalty and indirect cost amounts of \$2,000 and \$1,000/day, respectively.

5.1.1.1. Serial Merging Technique

In a network if there exist two activities $j_1, j_2 \in Acts$ such that j_1's unique successor is j_2 and j_2's unique predecessor is j_1, this network is said to be serial reducible through which activities j_1 and j_2 can be merged into an equivalent activity j_1'. The replaced activity j_1' will adopt the predecessors and successors from activities j_1 and j_2, respectively. Each time-cost alternative of j_1, $k_{j1} \in [1, m(j_1)]$, is then combined with each time-cost component of j_2, $k_{j2} \in [1, m(j_2)]$, forming the maximum $m(j_1') = m(j_1) \times m(j_2)$ number of candidate time-cost alternatives for the equivalent activity j_1'. Each of the time-cost

components of activity j_1', $k_{j1'} \in [1, m(j_1')]$, is achieved by summation of the times, $d_{k_{j1'}} = d_{k_{j1}} + d_{k_{j2}}$, and costs, $dc_{k_{j1'}} = dc_{k_{j1}} + dc_{k_{j2}}$, of the corresponding time-cost alternatives. In case of ties two similar but not identical approaches are implemented for the CS-Heuristic described in Section 5.1.3 and Section 5.1.4 and for the Mixed-Integer Linear Programming presented in Section 5.2.3. For CS-Heuristic, in case of a tie, all but one of the duplicate alternatives with $d_{k_{j1'}} = d_{(k+1)_{j1'}}$ and $dc_{k_{j1'}} = dc_{(k+1)_{j1'}}$ are eliminated. A dominance rule is also implemented to eliminate the dominated components as $k_{j1'} \succ (k+1)_{j1'}$, which indicates if $d_{k_{j1'}} \leq d_{(k+1)_{j1'}}$ while $dc_{k_{j1'}} \leq dc_{(k+1)_{j1'}}$ and one of these inequalities holds strictly, $(k+1)$ th alternative of the equivalent activity j_1' will be discarded. Whereas, for the Mixed-Integer Linear Programming in case of a tie, all but one of the duplicate alternatives with $d_{k_{j1'}} = d_{(k+1)_{j1'}}$ and $dc_{k_{j1'}} = dc_{(k+1)_{j1'}}$ are eliminated. A dominance rule is also implemented to eliminate the dominated components as $k_{j1'} \succ (k+1)_{j1'}$, which indicates if $d_{k_{j1'}} = d_{(k+1)_{j1'}}$ while $dc_{k_{j1'}} < dc_{(k+1)_{j1'}}$, $(k+1)$ th alternative of the equivalent activity j_1' will be eliminated. The flowchart of the proposed serial merging technique is illustrated graphically in Figure 5.6 of Section 5.1.3.

The procedure is best elucidated by a case example which is presented in Section 3.3.1. The original network shown in Figure 3.1 is serial reducible since Activity-1 and Activity-3 satisfy the above conditions. Activity-1 has only Activity-3 as its succeeding activity, and Activity-1 is the unique predecessor of Activity-3. It must be also pointed out that in this example the numerator of those activities with ID numbers greater than three are decreased by one. As shown in Figure 5.1, the equivalent activity, Activity-1, fetches the predecessors of the original Activity-1 (i.e., Start) and successors of the original Activity-3 (i.e., Activity-6 which is represented as Activity-5 in the reduced network).

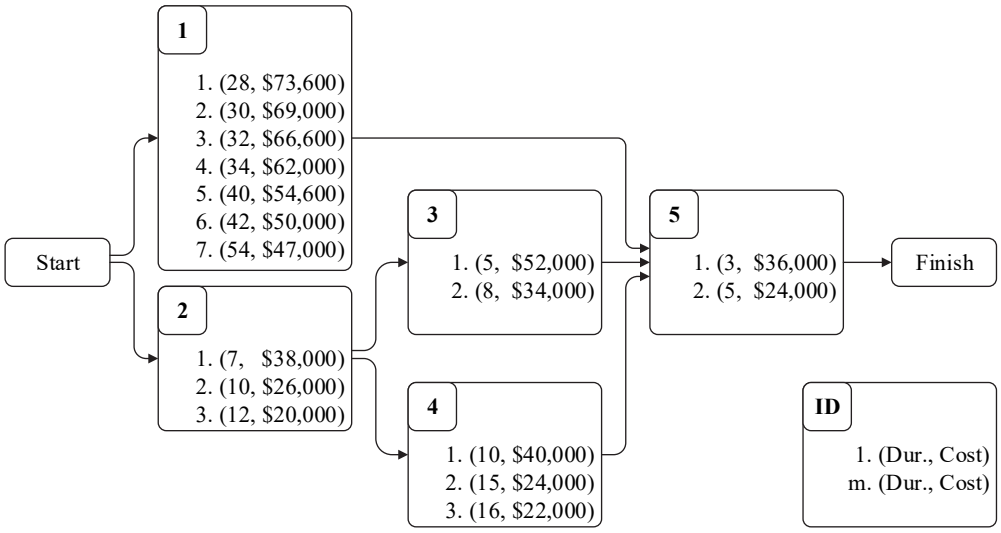


Figure 5.1 – Serial merge applied to the Case Example.

In the absence of the mode elimination rule mentioned earlier, the equivalent Activity-1 would have a maximum of $3 \times 3 = 9$ alternatives. However, combining the second mode of the original Activity-1 with the third mode of the original Activity-3 yields the same duration of 42 days achieved by the combination of the third mode of the original Activity-1 with the first mode of the original Activity-3. Since the cost of the first combination is added as $50,000 and the cost of the second combination is summed as $66,000, the second combination is discarded. In addition, summation of the third mode of the original Activity-1 with the second mode of the original Activity-3 yields a component with a duration of 46 days and direct cost of $59,000 which is dominated by the sixth mode of the equivalent Activity-1 with less duration and direct cost amounts. The solution space of the original network comprises 324 realizations; however, the solution space of the reduced network includes 252 different realizations. As is clear, the smaller solution space of the reduced network contributes to significantly faster computations.

5.1.1.2. Parallel Merging Technique

In a network if there exist two activities $j_1, j_2 \in Acts$ such that both j_1 and j_2 share a unique predecessor while sharing exactly the same set of successors, this network is said to be parallel reducible through which activities j_1 and j_2 can be merged into an equivalent activity j_1'. The replaced activity j_1' will adopt the single predecessor and the set of successors of the merged activities j_1 and j_2. Each time-cost alternative of j_1, $k_{j1} \in [1, m(j_1)]$, is then combined with each time-cost component of j_2, $k_{j2} \in [1, m(j_2)]$, forming the maximum $m(j_1') = m(j_1) + m(j_2) - 1$ number of candidate time-cost alternatives for the equivalent activity j_1'.Each of the time-cost components of activity j_1', $k_{j1'} \in [1, m(j_1')]$, is achieved by using an iterative scheme which initially starts by combining the all-normal modes of the j_1 and j_2 activities, i.e., $k_{j1} = m(j_1)$ and $k_{j2} = m(j_2)$. In each iteration of the pairwise combination, the larger duration is set as the duration of the new mode, $d_{k_{j1'}} = \max\{d_{k_{j1}}, d_{k_{j2}}\}$, while the cost of the new mode is calculated by adding the direct costs of the time-cost alternatives, $dc_{k_{j1'}} = dc_{k_{j1}} + dc_{k_{j2}}$. Thereafter, pairwise combination is carried out by incrementing the index of the driving time-cost component, $k_j \in \{k_{j1}, k_{j2}\}$, which had the greater amount of duration in the previous iteration, $\{k_j \mid d_{k_{j1'}} = d_{k_j}\}$. Thereby, the same procedure is repeated until the index of both the activities are promoted to their maximum amounts, viz., until all-crashed time-cost modes of the merged activities are experimented for combination, i.e., $k_{j1} = 1$ and $k_{j2} = 1$. Unlike serial merge, no elimination or dominance rule is incorporated since no ties occur in the above described procedure. The flowchart of the proposed parallel merging technique is illustrated graphically in Figure 5.5 of Section 5.1.3.

The procedure is best illustrated using a case example which is presented in Section 3.3.1. The original network shown in Figure 3.1 is parallel reducible since Activity-4 and Activity-5 satisfy the above conditions. Both the Activity-4 and Activity-5 has only Activity-2 as their preceding activity, and Activity-4 and Activity-5 only have Activity-6 in their set of successors which is common to both the activities 4 and 5. It must be also pointed out that in this example the numerator of those activities with ID numbers greater than five are decreased by one. As shown in Figure 5.2, the equivalent activity, Activity-4, fetches the unique predecessor of the original Activity-4/Activity-5 (i.e., Activity-2) as well as the successors of the original Activity-4/Activity-5 (i.e., Activity-6 which is represented as Activity-5 in the reduced network).

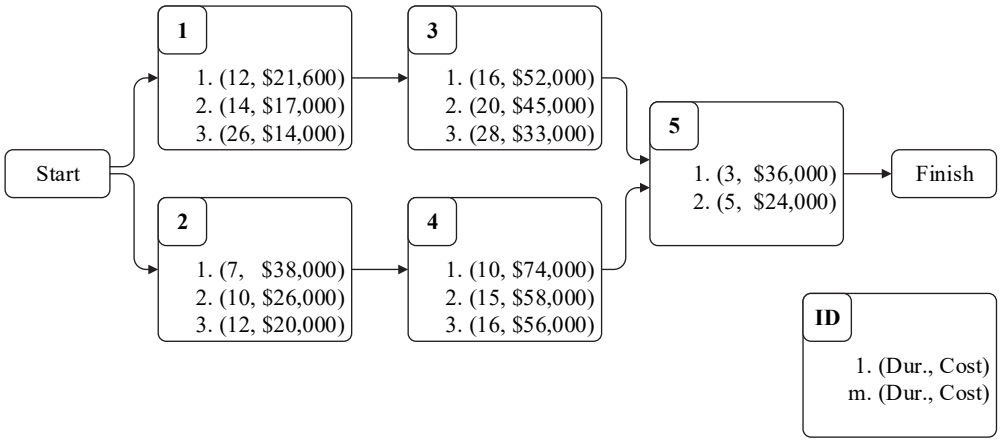


Figure 5.2 – Parallel merge applied to the Case Example.

The equivalent Activity-4 would have a maximum of $2+3-1=4$ alternatives. However, both the durations for both of the time-cost alternatives of the original Activity-4 (i.e., 5 and 8) are less than the duration of the all-crashed mode of the original Activity-5 (i.e., 10). Resultantly, the equivalent Activity-4 includes three time-cost modes with durations taken from the original Activity-5, while the costs are calculated by the summation of costs of the original Activity-5 with the least-cost mode of the original Activity-4 which is the second mode in this case. As

mentioned earlier, the solution space of the original network comprises 324 realizations; however, the solution space of the reduced network includes 162 different realizations. As is clear, the considerably smaller solution space of the reduced network contributes to significantly faster computations.

5.1.2. Partial-CPM Calculator

The computational bottleneck in solution of real-life-scale TCT problems is the fitness evaluation of the generated solutions due to the iterative and repetitive computation of the network using the critical path method. Computational complexities arise from the necessity to calculate the longest path of the network since any variation in selection of the alternatives modifies the project schedule. These incessant modifications require rescheduling the project for potential changes in its total cost and total duration amounts. It is obvious that any rescheduling process demands reanalyzing the network using the critical path method. Repetitive classical CPM calculations even with the modern computers is, therefore, not a convenient method for solution of DTCTP. In order to reduce the computational burden of the repeated CPM calculations, new approaches are implemented in the proposed CS-Heuristic. In the conventional CPM calculation module of the proposed method, where applicable, instead of applying forward and backward pass calculations, only the forward pass is applied and the total duration of the project is determined using the early finish date of the last activity on the project network. By means of this procedure, computation time is reduced for conditions where it is not necessary to carry out the complete CPM calculations which include computation of the late dates and floats. Original to this thesis, the proposed CS-Heuristic is also complemented with a unique CPM-esque approach to accelerate the solution process. A significant contribution of this thesis study is development of this unique network analyzer which is called the Partial-CPM calculator. Networks of the problems are calculated using this robust and efficient method which improves the overall computation time substantially.

Primarily, the distinction between a modified activity and an updated activity must be made as the first term refers to an activity with a modified setting of its time-cost alternatives; whereas, the second term refers to an activity with updated early/late start/finish dates. Rather than analyzing the whole network for changes, the proposed Partial-CPM technique examines only a small portion of the network as an alternative execution mode gets selected for an activity. In forward pass, this technique only updates early dates of the successors to the modified activity which have early-start dates smaller than early-finish date of the modified activity. In a similar fashion, the same adjustments are also applied to successors of the updated activities. In backward pass, Partial-CPM only updates late dates of the predecessors of the modified activity which have late-finish dates greater than late-start date of the modified activity. The same adjustments are also applied to predecessors of the updated activities. Following the forward pass and the backward pass calculations, total floats of all the updated activities are recalculated. To the best of author's knowledge, no such study exists in the literature. The main advantage of using the proposed Partial-CPM to DTCTP is that the CPM calculation of the entire network needs only be completed once, thereby the influence of modifications is analyzed by only experimenting a small fraction of the network.

Partial-CPM calculation procedure is described using a case example which is presented in Section 3.3.1. Figure 5.3 demonstrates the PERT chart for the all-crashed schedule. By uncrashing Activity-2 to its next option, its duration increases by three days (effective $DDiff$ amount) to 10 days.

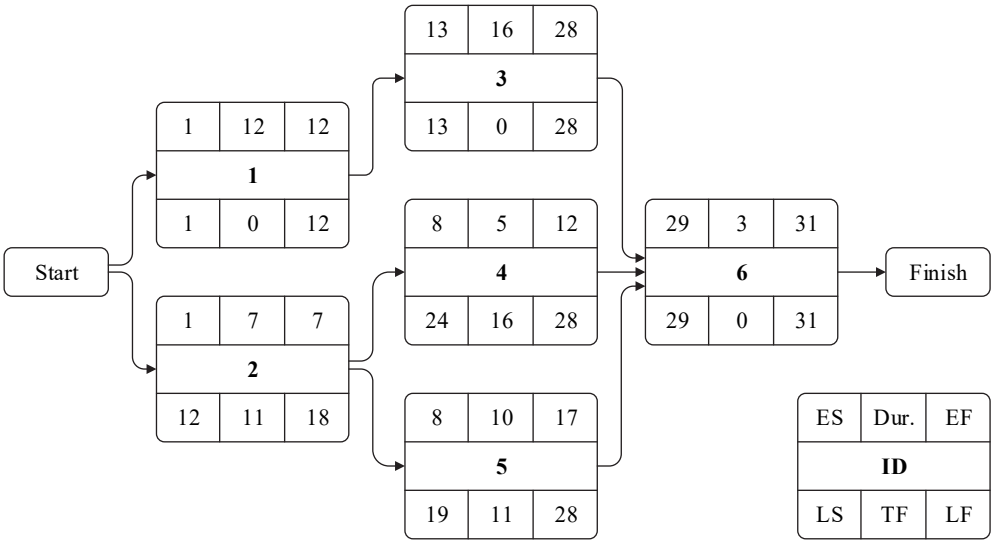


Figure 5.3 – PERT chart of all-crashed schedule for the Case Example.

Resultantly, its early-finish date increases from 7 to 10. According to Partial-CPM, in the first forward pass iteration, early dates of successors to Activity-2, i.e., Activity-4 and Activity-5 need to be updated if their early-start dates are smaller than 10. The current early-start dates of both Activity-4 and Activity-5 are shown to be 8, which satisfies the updating condition in forward pass of the Partial-CPM. Thus, this date is updated to $10+1=11$ and the early-finish date for each of these activities are updated by adding up their duration amounts to their early-start dates (Figure 5.4). Hence, early-finish dates of Activity-4 and Activity-5 are updated as 15 and 20, respectively. In the next iteration of forward pass, the successors to the updated activities, i.e., Activity-6 is evaluated. Since its early-start date of 29 is already greater than early-finish dates of both Activtiy-4 and Activity-5, and there exist no other successors to the updated activities, forward pass is terminated. In the first backward pass cycle, current late-finish date of the predecessors of the updated activities, i.e., Acitvity-2, is checked against late-start dates of both the updated activities Activity-4 and Activity-5. Backward pass is then terminated since late-finish date of 18 for Activity-2 is not greater than late-start dates of 24 and 19 for Activity-4 and Activtiy-5, respectively. Following the forward pass and the

backward pass calculations, total floats of all the updated activities are recalculated. It is shown that for this case example, Partial-CPM is capable of rescheduling the project by analyzing and updating only the dates/floats which are underscored in Figure 5.4. It is shown that the proposed technique, rather than rescheduling the entire network, is able to obtain the same rescheduled network by updating only 8 components (early dates, late dates, and floats) out of a total of 30.

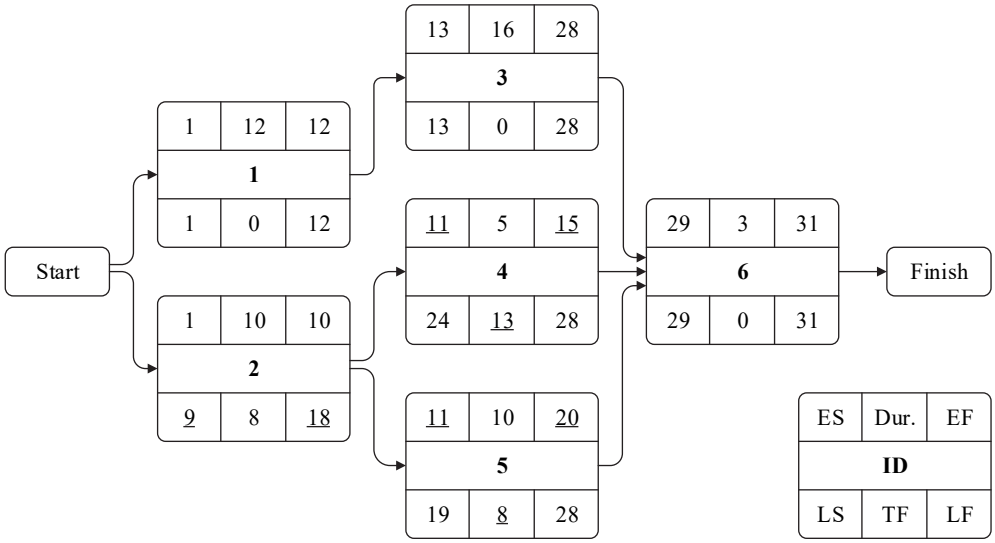


Figure 5.4 – PERT chart of the updated schedule for the Case Example.

5.1.3. CS-Heuristic for Deadline DTCTP

A new Cost-Slope Heuristic is presented herein for cost minimization and deadline discrete TCT problem. This method incorporates the project deadline, since, in practice there is a completion deadline stipulated in the contract for the majority of the projects. In realistic projects, delay penalties which are usually in the form of liquidated damages are applied in case the project duration exceeds the predetermined deadline and the incentives are the bonus payments made for each day saved from the specified deadline. Generally, there is a negative correlation

between the deadline and the complexity of the problem. That is, projects with larger predetermined thresholds are simpler than the ones with shorter deadlines. Regarding the above fact, the completion deadline of the practiced problems is calculated in the proposed heuristic algorithm as follows. Firstly, the all-normal schedule with the largest critical path length, $CPM_{\max}$, is calculated. Secondly, the all-crashed schedule with the shortest critical path, $CPM_{\min}$, is computed. Finally, the completion deadline is set to be equal to the average of the earliest allowable completion time of the project and the latest possible finishing time.

The first step of the CS-Heuristic involves serial/parallel merging of the network activities. Based on the techniques presented in Section 5.1.1.1 and Section 5.1.1.2 CS-Heuristic reduces the network of the problem to a simpler equivalent network. Thereafter it calculates the completion deadline based on the above described procedure. In the next step, CS-Heuristic calculates and records the cost-slopes (CS), and crash amounts (duration difference $DDiff$, and cost difference $CDiff$) for every time-cost option of all the activities. The cost-slopes are calculated according to Eq. (3.1) given in Section 3.2. All these options are then indexed as follows. Time-cost alternatives are sorted in descending order by $CDiff$ values then by CS values in descending order; thereby they are treated with an index named $UFF-index$. Similar to the previous stage, time-cost alternatives are sorted in descending order by CS values then by $DDiff$ values in ascending order, thereby an index called $UCS-index$ is assigned to each alternative. According to $UFF-index$, a component providing a greater cost saving is uncrashed first; in case of a tie, the component with a greater cost-slope rate is selected; if the tie is still not broken, the first component is selected. According to $UCS-index$, a component with a greater cost-slope rate is uncrashed first; in case of a tie, the component providing a smaller duration reduction is selected; if the tie is still not broken, the first component is selected.

Unlike earlier heuristic approaches, in lieu of starting from the all-normal schedule, $CPM_{\max}$, CS-Heuristic initiates from the all-crashed schedule, $CPM_{\min}$, by selecting the shortest/costliest time-cost alternative for the activities. Contrary to previous heuristics, in every cycle of CS-Heuristic, each new schedule is built upon an already non-dominated solution. Duration of this schedule is determined using CPM technique and the total cost is calculated by explicitly including the incentive and disincentive payments in the project cost formulation. Since CS-Heuristic starts from the all-crashed schedule, it is designed to optimize the solutions by performing an uncrashing scheme. The uncrashing scheme incorporates two different phases, namely, uncrash free-float (UFF) and uncrash cost-slope (UCS). Through UFF phase those activities with total floats greater than or equal to their effective (immediate available) crash amounts, $DDiff_{jk} \le TF_j$, are determined. The determined activities are then uncrashed to their next available option, by uncrashing one activity at a time, with respect to their $UFF-index$ values. After performing each UFF, CS-Heuristic uses Partial-CPM technique (Section 5.1.2) to update the modified activities. The UFF process is continued until either all the activities get fully uncrashed or their total floats become less than their effective crash amounts. It is after this step that the heuristic stores the current schedule as a solution. While each solution, Sol, obtained from the uncrashing phases are recorded in an archive called $Solutions$, an external repository, $Result$, is implemented to record the least-cost solution. Accordingly, the least-cost Sol of $Solutions$ is stored in $Result$, subsequently, the schedules stored in $Solutions$ are wiped out at the end of each iteration.

After the first schedule is generated and stored in $Result$, CS-Heuristic performs UCS process to generate new schedules unless the status of the unique solution stored in $Result$ is "Closed". Status of each solution which gets selected as a base for generation of new schedules, is switched to "Closed" at the end of each UCS phase, making it unavailable for re-selection. New schedules are generated based on the solutions that iteratively replace the single schedule recorded in $Result$, as

long as they are not "Closed". Through UCS phase those activities that are uncrashable, i.e., not fully uncrashed, are determined. The determined activities are then uncrashed to their next available option, by uncrashing one activity at a time, with respect to their $UCS-index$ values. After performing each UCS, CS-Heuristic implements the short procedure described in Section 5.1.2 to calculate the completion time of the obtained schedule. In case there exists no Sol stored in $Solutions$ with the same duration, CPM calculation is carried out which is then followed by the UFF process as mentioned earlier. It is after this step that the heuristic stores the current schedule as a solution. However, through UCS phase, if the same duration already exists in $Solutions$ within the same cycle, that schedule is discarded without being stored as a Sol in the archive. The UCS process is continued until all the determined uncrashable activities are uncrashed to their longer durations. After completing each UCS phase, the status of solution, $Sol \in Result$, which was selected at the start of the cycle is switched to "Closed". UCS and UFF phases are iteratively applied to the identified uncrashable activities, $\forall Sol \in (Solutions\ or\ Result) : status \neq Closed$, until obtaining a solution, all of the activities of which are uncrashed to their longest durations. CS-Heuristic terminates by returning the ultimate Sol stored in $Result$ with "Closed" status. The flowcharts of parallel merge, serial merge, and uncrash free-float (UFF) modules of the proposed CS-Heuristic are presented in Figure 5.5, Figure 5.6, and Figure 5.7, respectively. The pseudo-code of the proposed CS-Heuristic for cost minimization and deadline DTCTP is illustrated in Figure 5.8.

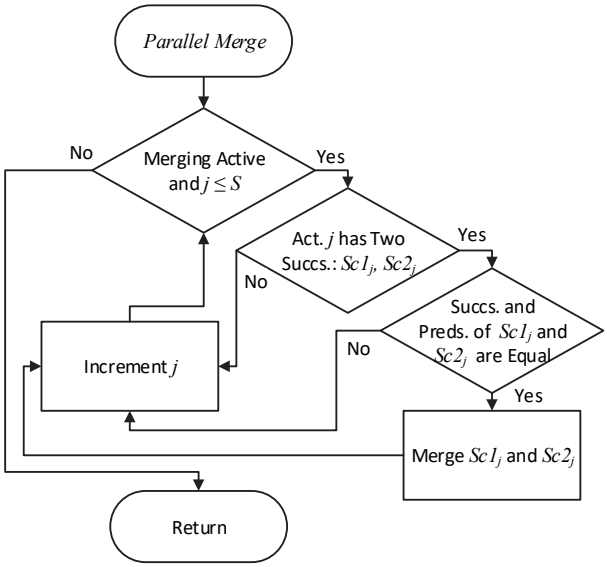


Figure 5.5 – Flowchart of the proposed parallel merging technique.

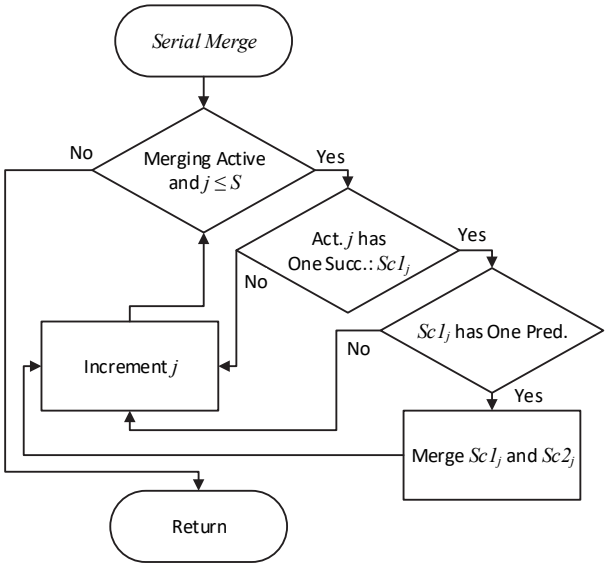


Figure 5.6 – Flowchart of the proposed serial merging technique.

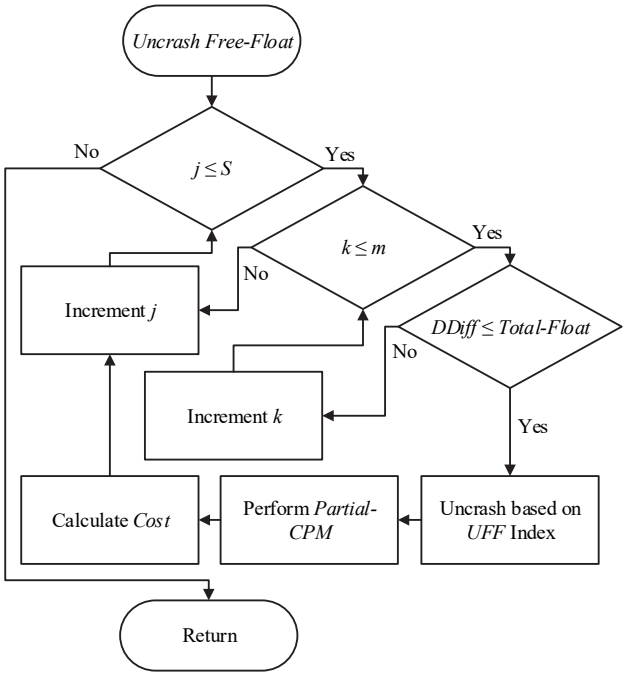


Figure 5.7 – Flowchart of the uncrash free-float module of the proposed CS-Heuristic.

```
Begin;
    For ∀ File in Directory
        For ∀ j ∈ [1, S]
            For ∀ k ∈ [1, m]
                Retrieve Values;
            End;
        End;
        For ∀ j ∈ Acts
            Calculate Predecessors;
        End;
        Sort Acts;
        If Perform Parallel Merge = True
            For ∀ j ∈ Acts
                If j has 2 Successors
                    If Successors of Successors to j ∧ Predecessors of Successors to j are same
                        Merge Successors to j;
                    End;
                End;
            End;
        End;
        If Perform Serial Merge = True
            For ∀ j ∈ Acts
```

Figure 5.8 – Pseudo-code of the proposed CS-Heuristic for deadline DTCTP.

```
            If j has 1 Successor
                If Successor to j has 1 Predecessor
                    Merge j and its Successor;
                End;
            End;
        End;
    End;
    Calculate Project Deadline;
    For ∀ j ∈ Acts
        For ∀ k ∈ Modes
            Calculate CS, DDiff, CDiff;
            Sort and Index by CDiff then by CS for Uncrash Free-Float;
            Sort and Index by CS then by DDiff for Uncrash Cost-Slope;
        End;
        Select 1st Mode;
    End;
    Calculate CPM;
    Calculate Dur/Cost;
    While ∃ j ∈ Acts: DDiff ≤ Total-Float
        Add to Uncrashables;
        Perform Uncrash Free-Float;
        Perform Partial-CPM;
        Calculate Cost;
    Break;
    Store Result;
    While Result ≠ Closed
        For ∀ Act ∈ Result: Fully-Uncrashed ≠ True
            Add to Uncrashables;
        End;
        Perform Uncrash Cost-Slope;
        Perform Partial-CPM;
        Calculate Dur;
        If ∃ Sol ∈ Solutions: Sol.Dur = Dur
            ;
        End;
        Else
            Calculate CPM;
            Calculate Cost;
            While ∃ j ∈ Acts: DDiff ≤ Total-Float
                Add to Uncrashables;
                Perform Uncrash Free-Float;
                Perform Partial-CPM;
                Calculate Cost;
            Break;
            Store Sol in Solutions;
        End;
        Close Result;
```

Figure 5.8 – Pseudo-code of the proposed CS-Heuristic for deadline DTCTP (*Continued*).

```
            If ∃ Sol ∈ Solutions: Sol.Cost ≤ Result.Cost
                Result = Sol;
            End;
        Break;
        Return Result;
    End;
End;
```

Figure 5.8 – Pseudo-code of the proposed CS-Heuristic for deadline DTCTP (*Continued*).

The proposed CS-Heuristic is described by a case example which is presented in Section 3.3.1. The original network shown in Figure 3.1 is used to elucidate *UFF* and *UCS* schemes of the CS-Heuristic. For the all-crashed schedule, the project duration is 31 days and the total cost is $270,600 as shown in Table 5.1.

Table 5.1 – Candidate solutions found by CS-Heuristic for deadline DTCTP.

# of Schedule	Continue from	Activity	Crash Level	Uncrash	Dur. (day)	Direct Cost ($)	Indirect Cost ($)	Total Cost ($)
1			0	Crashed	31	239,600	31,000	270,600
1.i	1	4	0	M1 to M2	31	221,600	31,000	252,600
1.ii	1.i	5	0	M1 to M2	31	205,600	31,000	236,600
1.iii	1.ii	2	0	M1 to M2	31	193,600	31,000	224,600
1.iv	1.iii	2	0	M2 to M3	31	187,600	31,000	218,600
1.v	1.iv	5	0	M2 to M3	31	185,600	31,000	216,600
2	1.v	6	1	M1 to M2	33	173,600	33,000	206,600
-	1.v	1	1	M1 to M2	33	-	-	-
3	1.v	3	1	M1 to M2	35	178,600	35,000	213,600
4	2	1	2	M1 to M2	35	169,000	35,000	204,000
5	3	3	2	M1 to M2	37	166,600	37,000	203,600
6	4	3	3	M1 to M2	39	162,000	39,000	201,000
7	4	1	3	M2 to M3	47	166,000	47,000	213,000
8	5	1	4	M1 to M2	39	162,000	39,000	201,000
9	**5**	**3**	**4**	**M2 to M3**	**45**	**154,600**	**45,000**	**199,600**
10	6	3	5	M2 to M3	47	150,000	47,000	201,000*
11	6	1	5	M2 to M3	51	159,000	51,000	210,000
12	9	1	6	M1 to M2	47	150,000	47,000	201,000*

*Exceeds deadline by 2 days

In the Schedule-1 which consists of crashed modes, activities 2, 4, and 5 are non-critical activities. According to *UFF* , among the non-critical activities, Activity-4 with a *CDiff* of $18,000 provides the greatest cost saving as shown in Table 5.2, and is uncrashed first, to its second mode (M2).

Table 5.2 – Cost-slopes, *DDiff* s and *CDiff* s of crash modes.

Activity	Crash Mode	Cost-slope ($/Day)	*DDiff* (day)	*CDiff* ($)
1	M1 to M2	2,300	2	4,600
1	M2 to M3	250	12	3,000
2	M1 to M2	4,000	3	12,000
2	M2 to M3	3,000	2	6,000
3	M1 to M2	1,750	4	7,000
3	M2 to M3	1,500	8	12,000
4	M1 to M2	6,000	3	18,000
5	M1 to M2	3,200	5	16,000
5	M2 to M3	2,000	1	2,000
6	M1 to M2	6,000	2	12,000

The same procedure is applied to Activity-5 which is uncrashed to its second mode (M2), similarly, Activity-2 is uncrashed to its second mode (M2), as well. In schedules Schedule-1.iv and Schedule-1.v, Activity-2 and Activity-5 are uncrashed to their third option (M3), respectively. The resulting Schedule-1.v has a duration of 31 days and a total cost of $216,600 (Table 5.1) which is recorded in *Result* as the first *UFF* cycle terminates. Following *UFF* phase, five uncrashing options could be identified for *UCS* including Activity-1: M1 to M2, Activity-1: M2 to M3, Activity-3: M1 to M2, Activity-3: M2 to M3, and Activity-6: M1 to M2. However, M2 to M3 crash options of Activity-1 and Activity-3 are excluded from the uncrashables, since, only the effective (immediate available) crash options are considered throughout *UCS* phase. Therefore, succeeding *UFF* phase, Activity-6 with largest cost-slope of $6,000/day is uncrashed through the *UCS* process and stored as Schedule-2 in *Solutions* . The next least-cost-slope activity is Activity-1, however, uncrashing it to its second mode (M2) results in a solution with a duration

of 33 days which already exists as Schedule-2 in *Solutions* ; hence, this schedule is discarded. The final uncrashable activity in this cycle of *UCS* is Activity-3, which is uncrashed to its second mode (M2). For none of the schedules of the first *UCS* cycle, *UFF* process was applied since there were no uncrashable non-critical activities in the network. As mentioned earlier, status of each solution which gets selected as a base for generation of new schedules, is switched to "Closed" at the end of each *UCS* phase. Accordingly, at the end of the first *UCS* cycle the status of Schedule-1.v is changed to "Closed" and the solution in *Result* is replaced by Schedule-2 having the least total cost found so far. Thereafter, the components of the *Solutions* are erased. These iterative cycles are repeated for the remainder of the time-cost options. Since Schedule-10 and Schedule-12 exceed the deadline of 45 days by two days, the overall cost is increased from $197,000 to $201,000 for these solutions. Finally, the CS-Heuristic terminates by returning the last *Sol* recorded in *Result* – Schedule-9, highlighted in boldface in Table 5.1 – with a duration of 45 days and a total cost of $199,600.

5.1.4. CS-Heuristic for Pareto front DTCTP

Similar to the heuristic approach presented in Section 5.1.3, a new Cost-Slope Heuristic is presented herein for Pareto front discrete TCT problem. This method also incorporates the project deadline, since, in practice there is a completion deadline stipulated in the contract for the majority of the projects. In realistic projects, delay penalties which are usually in the form of liquidated damages are applied in case the project duration exceeds the predetermined deadline and the incentives are the bonus payments made for each day saved from the specified deadline. Generally, there is a negative correlation between the deadline and the complexity of the problem. That is, projects with larger predetermined thresholds are simpler than the ones with shorter deadlines. Regarding the above fact, the completion deadline of the practiced problems is calculated in the proposed heuristic algorithm as follows. Firstly, the all-normal schedule with the largest

critical path length, $CPM_{\max}$, is calculated. Secondly, the all-crashed schedule with the shortest critical path, $CPM_{\min}$, is computed. Finally, the completion deadline is set to be equal to the average of the earliest allowable completion time of the project and the latest possible finishing time.

The first step of the CS-Heuristic involves serial/parallel merging of the network activities. Based on the techniques presented in Section 5.1.1.1 and Section 5.1.1.2 CS-Heuristic reduces the network of the problem to a simpler equivalent network. Thereafter it calculates the completion deadline based on the above described procedure. In the next step, CS-Heuristic calculates and records the cost-slopes (CS), and crash amounts (duration difference $DDiff$, and cost difference $CDiff$) for every time-cost option of all the activities. The cost-slopes are calculated according to Eq. (3.1) given in Section 3.2. All these options are then indexed as follows. Time-cost alternatives are sorted in descending order by $CDiff$ values then by CS values in descending order; thereby they are treated with an index named $UFF-index$. Similar to the previous stage, time-cost alternatives are sorted in descending order by CS values then by $DDiff$ values in ascending order, thereby an index called $UCS-index$ is assigned to each alternative. According to $UFF-index$, a component providing a greater cost saving is uncrashed first; in case of a tie, the component with a greater cost-slope rate is selected; if the tie is still not broken, the first component is selected. According to $UCS-index$, a component with a greater cost-slope rate is uncrashed first; in case of a tie, the component providing a smaller duration reduction is selected; if the tie is still not broken, the first component is selected.

Unlike earlier heuristic approaches, in lieu of starting from the all-normal schedule, $CPM_{\max}$, CS-Heuristic initiates from the all-crashed schedule, $CPM_{\min}$, by selecting the shortest/costliest time-cost alternative for the activities. Contrary to previous heuristics, in every cycle of CS-Heuristic, each new schedule is built upon

an already non-dominated solution. Duration of this schedule is determined using CPM technique and the total cost is calculated by explicitly including the incentive and disincentive payments in the project cost formulation. Since CS-Heuristic starts from the all-crashed schedule, it is designed to optimize the solutions by performing an uncrashing scheme. The uncrashing scheme incorporates two different phases, namely, uncrash free-float (UFF) and uncrash cost-slope (UCS). Through UFF phase those activities with total floats greater than or equal to their effective (immediate available) crash amounts, $DDiff_{jk} \leq TF_j$, are determined. The determined activities are then uncrashed to their next available option, by uncrashing one activity at a time, with respect to their $UFF-index$ values. After performing each UFF, CS-Heuristic uses Partial-CPM technique (Section 5.1.2) to update the modified activities. The UFF process is continued until either all the activities get fully uncrashed or their total floats become less than their effective crash amounts. It is after this step that the heuristic stores the current schedule as a solution. While each solution, Sol, obtained from the uncrashing phases are recorded in an archive called $Solutions$, an external repository, $Pareto$, is implemented to record the non-dominated solutions. Accordingly, the non-dominated Sols of $Solutions$ are copied to $Pareto$ and the solutions stored in $Pareto$ are sorted according to their durations in ascending order; subsequently, the schedules stored in $Solutions$ are wiped out at the end of each iteration.

After the first set of non-dominated schedules are generated and stored in $Pareto$, CS-Heuristic performs UCS process on the first non-dominated solution – with the least duration – archived in $Pareto$ to generate new schedules unless the status of the selected solution is "Closed". Status of each solution which gets selected as a base for generation of new schedules, is switched to "Closed" at the end of each UCS phase, making it unavailable for re-selection. New schedules are generated based on the non-dominated solutions which are already stored in $Pareto$, as long as they are not "Closed". Through UCS phase those activities that are uncrashable, i.e., not fully uncrashed, are determined. The determined activities are then

uncrashed to their next available option, by uncrashing one activity at a time, with respect to their $UCS-index$ values. After performing each UCS, CS-Heuristic implements the short procedure described in Section 5.1.2 to calculate the completion time of the obtained schedule. In case there exists no Sol stored in $Solutions$ with the same duration, CPM calculation is carried out which is then followed by the UFF process as mentioned earlier. It is after this step that the heuristic stores the current schedule as a solution in $Solutions$. However, through UCS phase, if the same duration already exists in $Solutions$ within the same cycle, that schedule is discarded without being stored as a Sol in the archive. The UCS process is continued until all the determined uncrashable activities are uncrashed to their longer durations. After completing each UCS phase, the status of solution, $Sol \in Pareto$, which was selected at the start of the cycle is switched to "Closed" and the dominated solutions are removed from $Solutions$. UCS and UFF phases are iteratively applied to any of the uncrashable activities of the non-dominated solutions, $\forall Sol \in (Solutions\ or\ Pareto) : status \neq Closed$, until obtaining a solution, all of the activities of which are uncrashed to their longest durations. CS-Heuristic terminates by returning the ultimate non-dominated Sols stored in $Pareto$ with "Closed" statuses. The flowcharts of parallel merge, serial merge, and uncrash free-float (UFF) modules of the proposed CS-Heuristic are presented in Figure 5.5, Figure 5.6, and Figure 5.7 of Section 5.1.3, respectively. The proposed CS-Heuristic approach for Pareto front DTCTP is graphically explained as a flowchart in Figure 5.9 and the pseudo-code of this method is demonstrated in Figure 5.10.

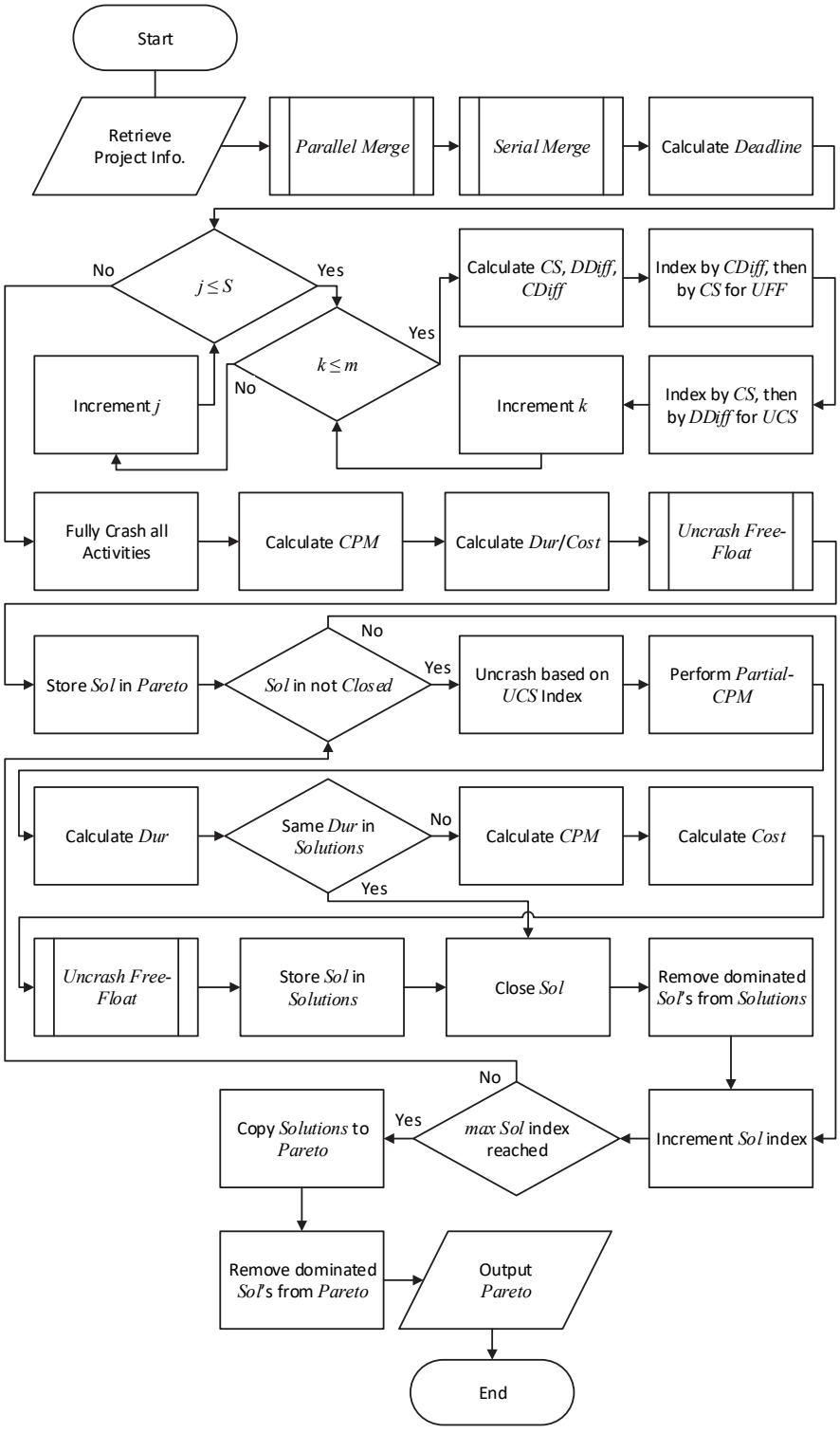


Figure 5.9 – Flowchart of the proposed CS-Heuristic for Pareto front DTCTP.

```
Begin;
    For ∀ File in Directory
        For ∀ j ∈ [1, S]
            For ∀ k ∈ [1, m]
                Retrieve Values;
            End;
        End;
        For ∀ j ∈ Acts
            Calculate Predecessors;
        End;
        Sort Acts;
        If Perform Parallel Merge = True
            For ∀ j ∈ Acts
                If j has 2 Successors
                    If Successors of Successors to j ∧ Predecessors of Successors to j are same
                        Merge Successors to j;
                    End;
                End;
            End;
        End;
        If Perform Serial Merge = True
            For ∀ j ∈ Acts
                If j has 1 Successor
                    If Successor to j has 1 Predecessor
                        Merge j and its Successor;
                    End;
                End;
            End;
        End;
        Calculate Project Deadline;
        For ∀ j ∈ Acts
            For ∀ k ∈ Modes
                Calculate CS, DDiff, CDiff;
                Sort and Index by CDiff then by CS for Uncrash Free-Float;
                Sort and Index by CS then by DDiff for Uncrash Cost-Slope;
            End;
            Select 1st Mode;
        End;
        Calculate CPM;
        Calculate Dur/Cost;
        While ∃ j ∈ Acts: DDiff ≤ Total-Float
            Add to Uncrashables;
            Perform Uncrash Free-Float;
            Perform Partial-CPM;
            Calculate Cost;
        Break;
        Store Sol in Pareto;
```

Figure 5.10 – Pseudo-code of the proposed CS-Heuristic for Pareto front DTCTP.

```
        While ∃ Sol ∈ Pareto: Sol ≠ Closed
            For ∀ Act ∈ Sol: Fully-Uncrashed ≠ True
                Add to Uncrashables;
            End;
            Perform Uncrash Cost-Slope;
            Perform Partial-CPM;
            Calculate Dur;
            If ∃ Sol ∈ Solutions: Sol.Dur = Dur
                ;
            End;
            Else
                Calculate CPM;
                Calculate Cost;
                While ∃ j ∈ Acts: DDiff ≤ Total-Float
                    Add to Uncrashables;
                    Perform Uncrash Free-Float;
                    Perform Partial-CPM;
                    Calculate Cost;
                Break;
                Store Sol in Solutions;
            End;
            Close Sol;
            For ∀ Sol ∈ Solutions: Non-dominated ≠ True
                Remove Sol;
            End;
            Copy Solutions to Pareto;
        Break;
        For ∀ Sol ∈ Pareto: Non-dominated ≠ True
            Remove Sol;
        End;
        Return Pareto;
    End;
End;
```

Figure 5.10 – Pseudo-code of the proposed CS-Heuristic for Pareto front DTCTP (*Continued*).

The proposed CS-Heuristic is explained using the same case problem which is presented in Section 3.3.1. The original network shown in Figure 3.1 is used to elucidate *UFF* and *UCS* processes of the CS-Heuristic. For the all-crashed schedule, the project duration is 31 days and the total cost is $270,600 as shown in Table 5.3.

Table 5.3 – Candidate solutions found by CS-Heuristic for deadline DTCTP.

# of Schedule	Continue from	Activity	Crash Level	Uncrash	Dur. (day)	Direct Cost ($)	Indirect Cost ($)	Total Cost ($)
1			0	Crashed	31	239,600	31,000	270,600
1.i	1	4	0	M1 to M2	31	221,600	31,000	252,600
1.ii	1.i	5	0	M1 to M2	31	205,600	31,000	236,600
1.iii	1.ii	2	0	M1 to M2	31	193,600	31,000	224,600
1.iv	1.iii	2	0	M2 to M3	31	187,600	31,000	218,600
1.v	**1.iv**	**5**	**0**	**M2 to M3**	**31**	**185,600**	**31,000**	**216,600**
2	**1.v**	**6**	**1**	**M1 to M2**	**33**	**173,600**	**33,000**	**206,600**
-	1.v	1	1	M1 to M2	33	-	-	-
3	1.v	3	1	M1 to M2	35	178,600	35,000	213,600
4	**2**	**1**	**2**	**M1 to M2**	**35**	**169,000**	**35,000**	**204,000**
5	**3**	**3**	**2**	**M1 to M2**	**37**	**166,600**	**37,000**	**203,600**
6	**4**	**3**	**3**	**M1 to M2**	**39**	**162,000**	**39,000**	**201,000**
7	4	1	3	M2 to M3	47	166,000	47,000	213,000
8	5	1	4	M1 to M2	39	162,000	39,000	201,000
9	**5**	**3**	**4**	**M2 to M3**	**45**	**154,600**	**45,000**	**199,600**
10	6	3	5	M2 to M3	47	150,000	47,000	201,000*
11	6	1	5	M2 to M3	51	159,000	51,000	210,000
12	9	1	6	M1 to M2	47	150,000	47,000	201,000*

*Exceeds deadline by 2 days

In the Schedule-1 which consists of crashed modes, activities 1, 3, and 6 are on the critical path. According to *UFF* , among the non-critical activities, Activity-4 with a *CDiff* of $18,000 provides the greatest cost saving as shown in Table 5.2 given in Section 5.1.3. Thus, Activity-4 is uncrashed first, to its second mode (M2). The same procedure is applied to Activity-5 which is uncrashed to its second mode (M2), similarly, Activity-2 is also uncrashed to its second mode (M2). In schedules Schedule-1.iv and Schedule-1.v, Activity-2 and Activity-5 are uncrashed to their third option (M3), respectively. The resulting Schedule-1.v has a duration of 31 days and a total cost of $216,600 (Table 5.3) which is recorded in *Pareto* as the first *UFF* cycle terminates. Following *UFF* phase, five uncrashing options could be identified for *UCS* including Activity-1: M1 to M2, Activity-1: M2 to M3, Activity-3: M1 to M2, Activity-3: M2 to M3, and Activity-6: M1 to M2. However, M2 to M3 crash options of Activity-1 and Activity-3 are excluded from the

uncrashables, since, only the effective (immediate available) crash options are considered throughout *UCS* process. Therefore, succeeding *UFF* phase, Activity-6 with largest cost-slope of $6,000/day is uncrashed to its second mode (M2) in the course of *UCS* phase and stored as Schedule-2 in *Solutions*. The next least-cost-slope activity is Activity-1, however, uncrashing it to its second mode (M2) results in a solution with a duration of 33 days which already exists as Schehule-2 in *Solutions*; hence, this schedule is discarded. The final uncrashable activity in this cycle of *UCS* is Activity-3, which is uncrashed to its second mode (M2). For none of the schedules of the first *UCS* cycle, *UFF* process was applied since there were no uncrashable non-critical activities in the network. At the end of the first *UCS* cycle the non-dominated *Sol* in *Solutions*, i.e., Schedule-2, is copied to *Pareto*. As mentioned earlier, status of each solutions which gets selected as a base for generation of new schedules, is switched to "Closed" at the end of each *UCS* phase. Accordingly, the status of Schedule-1.v is then changed to "Closed" and the components of the *Solutions* are erased. In the second cycle of *UCS*, the least-cost schedule is copied to now blank *Solutions*, unless its status is "Closed". In this case, Schedule-2 is copied to the repository to carry out *UCS*. Based on this schedule, Schedule-4 is generated by uncrashing Activity-1 to its second mode (M2). Next, Activity-3 is uncrashed to its second mode (M2). Since Activity-3 is already uncrashed to its second mode (M2) in the preceding cycle of *UCS*, it is now possible for this activity to be uncrashed to its third mode (M3) in the third cycle of *UCS*, which results in Schedule-6 as shown in Table 5.3. The final uncrashable activity in this cycle of *UCS* is Activity-1. The resulting schedule has a duration of 47 and total cost of $213,000, which is already dominated by Schedule-6 stored in *Solutions*. In fourth cycle of *UCS*, Schedule-5 is used as the base schedule. Uncrashing Activity-1 with the largest *CS* results in exactly the same solution already recorded in *Pareto* as Schedule-6. The final uncrashable activity of cycle five is Activity-3 which is uncrashed to its third option (M3). In the sixth cycle of *UCS*, Schedule-10 is obtained, completion time of which exceed the deadline by two days. The next schedule achieved in this cycle include

Schedule-11 as shown in Table 5.3. Through the last cycle of *UCS* , Schedule-12 is obtained which also exceeds the deadline of 45 days by two days, with an overall cost of $201,000. Finally, the CS-Heuristic terminates by returning the last non-dominated *Sol* s recorded in *Pareto* which includes Schedule-1.v, Schedule-2, Schedule-4, Schedule-5, Schedule-6, and Schedule-9. The highlighted schedules shown in boldface in Table 5.3 constitute the final set of non-dominated solutions found by the CS-Heuristic for the case problem.

5.2. Computational Experiments of CS-Heuristic

5.2.1. Generation of New Sets of Instances

It is mentioned earlier that no standard test-bed is available in the literature that could be considered to be as complex as the real-life projects. Therefore, several random DTCTP instances are generated systematically in this thesis to measure the performances and evaluate the capabilities of the proposed optimization algorithms. There exist numerous random network generators proposed within the literature including ProGen/Max (Schwindt, 1995), RanGen (Demeulemeester, Dodin, and Herroelen, 1993; Demeulemeester, Vanhoucke, and Herroelen, 2003), DAGEN (Agrawal, Elmaghraby, and Herroelen, 1996), and RanGen2 (Vanhoucke, Coelho, Debels, Maenhout, and Tavares, 2008). Except for DAGEN, all the other network generators generate AoN type of networks. Since this thesis uses AoN notation system, DAGEN generator is dismissed. On the other hand, it is shown in the study by Demeulemeester et al. (2003) that RanGen is more advantageous compared with other earlier instance generators including ProGen/Max and DAGEN and topological sorting of the networks generated by means of RanGen are more complicated. In a more recent study, Vanhoucke et al. (2008) show their RanGen2 generator outperforms ProGen (Kolisch, Sprecher, and Drexl, 1995), RanGen, and RiskNet (Tavares, 1999) generators with respect to the total amount of networks generated. RanGen2 is capable of generating networks with complexities on par

with the original RanGen. In addition, RanGen2 is shown to best the original RanGen since it starts from a larger pool of possible networks and is able to generate different networks, whereas, RanGen is able to generate exactly one network per run. Accordingly, in this thesis, RanGen2 random network generator (Figure 5.11) is implemented to generate strongly random activity networks.

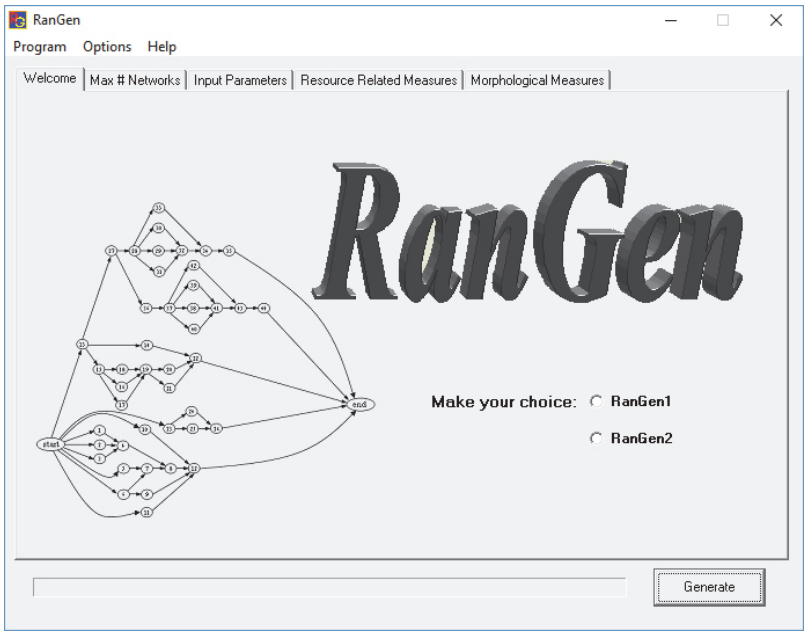


Figure 5.11 – RanGen2 Interface.

RanGen2 is actually a modified version of RanGen which incorporates alternative topological indicators. RanGen2 aims at generating different topological structures using predefined values for I_1, I_2, I_3, I_4, I_5, and I_6 parameters. These parameters are defined as follows (Vanhoucke et al., 2008):

- I_1: Network size indicator: Number of activities;
- I_2: Serial/parallel indicator: Closeness of a network to a serial ($I_2 = 1$, a chain of activities) or parallel ($I_2 = 0$, no precedence relations) graph;

- I_3: Activity distribution indicator: Distribution of the activities over the network ($I_3 = 0$, uniform distribution of activities);
- I_4: Short precedence relations indicator: Presence of short precedence relations;
- I_5: Long precedence relations indicator: Presence of long precedence relations;
- I_6: Topological float: Topological float of activities in the network ($I_6 = 0$, all activities have topological float of zero).

New sets of DTCT problems are created as txt files by combining complex networks – generated by means of RanGen2 – with realistic sets of time-cost alternatives – developed by means of a code written in C# based on Eq. (5.1) given below. Characteristics of the generated random instances can be summarized as follows:

- I_1: Number of activities include five levels of 50, 100, 200, 500, and 990;
- I_2: Serial/Parallel factor is varied from 0.2 (almost parallel) to 0.8 (almost serial) in steps of 0.2, i.e., 0.2, 0.4, 0.6, and 0.8;
- I_3 to I_6: These parameters are drawn randomly from $\mathrm{U}(0,1)$ in order to make a balanced representation of different network properties;
- m: Number of time-cost alternatives include three fixed levels of 3, 6, and 9.

For each parameter level combination, i.e., I_1, I_2, and m, ten replications of networks are generated leading to a total of $5 \times 4 \times 3 \times 10 = 600$ instances. In doing so, a wide range of complexity with respect to the topological structure is captured. The reason behind generation of instances with 990 activities is simply because RanGen2 is unable to generate samples for greater I_1 values. Each of the project sizes can only have a fixed number of alternatives as 3, 6, or 9 which are generated by means of non-increasing convex time-cost functions using the procedure described in Akkan et al. (2005). Fixed number of time-cost alternatives are

assigned for the activities since according to Vanhoucke and Debels (2007), such problems are more difficult to solve than the instances that involve a random number selected from a predefined range. As stated by Vanhoucke and Debels (2007), variation in the number of modes through the activities reduces the complexity of the problems.

For each activity, the time-cost alternatives are generated as follows: First the number of modes (m) is decided for all the activities, i.e., in an instance all the activities include either 3, 6, or 9 alternatives. Then, the durations of these modes are randomly sampled from $\mathrm{DU}(1,54)$ (i.e., discrete uniform distribution with parameters 1 and 54) as follows: The range 1-54 is divided into m number of intervals. Duration for the normal mode is selected randomly from the last interval, and the duration of each of the succeeding alternatives are assigned randomly from the corresponding preceding intervals; viz., duration for the crashed mode is selected from the first interval using a random scheme. After determining all the duration amounts, the costs of the modes are generated sequentially, starting with the normal mode (least-cost alternative) by multiplying its duration by a value (dc_{Int}) randomly sampled from $\mathrm{DU}(500,2000)$. Direct costs of the subsequent alternatives are generated using Eq. (5.1).

$$\begin{aligned} & dc_{jk} = dc_{j(k-1)} + CS_{jk} \times dc_{Int} \times (d_{j(k-1)} - d_{jk}) \\ & \forall\, j = \{1,\ldots,S\} \quad , \quad \forall\, k = \{2,\ldots,m(j)\} \end{aligned} \tag{5.1}$$

where dc_{jk} is the direct cost of the k th alternative of the j th activity; $dc_{j(k-1)}$ is the direct cost of the $(k-1)$ th alternative of th j th activity; CS_{jk} is the cost-slope percentage of k th option of j th activity which is randomly sampled from the uniform distribution $\mathrm{U}(0.1,0.5)$ as follows: The range 10%-50% is divided into $(m-1)$ number of intervals, CS_{jk} for the k th alternative is selected randomly from the first interval and CS_{jk} of each of the succeeding alternatives are selected

randomly from the corresponding subsequent intervals which secures an incremental cost-slope pattern; dc_{Int} is a value randomly selected from $\mathrm{DU}(500, 2000)$; $d_{j(k-1)}$ and d_{jk} are durations of $(k-1)$th and kth options, respectively. According to Eq. (5.1), mutually incomparable modes are generated for the activities. Besides, the implemented incremental cost-slopes scheme is capable of reflecting the realistic decline in productivity rate, since, activities are usually crashed by overstaffing or by working overtime. As a result of this, the realistic crash options lead to a convex solution space. The generated instances, akin to real-life projects, have a convex time-cost relationship for the projects as a whole and for each of its activities.

Completion deadlines are also included in the new sets of instances, since, in practice there is a completion deadline stipulated in the contract for the majority of the projects. In realistic projects, delay penalties which are usually in the form of liquidated damages are applied in case the project duration exceeds the predetermined deadline. Generally, there is a negative correlation between the deadline and the complexity of the problem. That is, projects with larger predetermined thresholds are simpler than the ones with shorter deadlines. Regarding the above fact, the completion deadlines of the generated instances are calculated as follows. Firstly, the all-normal schedule with the largest critical path length, $CPM_{\max}$, is calculated. Secondly, the all-crashed schedule with the shortest critical path, $CPM_{\min}$, is computed. Finally, the completion deadline is set to be equal to the average of the earliest allowable completion time of the project and the latest possible finishing time. The indirect cost rate is set as \$1000/day for all the instances and a twofold of the indirect cost is used as the amount of the delay penalty, i.e., \$2,000/day. The order of numbering for each setting of the generated instances is illustrated in Table 5.4. As presented in Section 5.2.3 and Section 5.2.5.4, behavior of the proposed exact and heuristic/meta-heuristic methods are dissimilar for each corresponding setting of the parameters. According to the findings of Section 5.2.3, the nature of mixed-integer linear programming technique

tends to be more suitable for pseudo-serial networks with greater I_2 values that include smaller number of time-cost modes. Contrary to mixed-integer linear programming technique, results of Section 5.2.5.4 reveal that the nature of the proposed heuristic and meta-heuristic approaches are more suitable for solution of pseudo-parallel networks with smaller I_2 values that comprise larger number of time-cost alternatives. Thus, problems 21 to 30 are considered to be the most complicated, and the problems 91 to 100 are regarded as the simplest problems for the proposed exact algorithm. The contrary, problems 91 to 100 are experienced to be the most complex, while, problems 21 to 30 to be the least complicated instances for the proposed heuristic/meta-heuristic solution procedures.

Table 5.4 – Complexity of the generated instances.

	I_2			
# of Modes	**0.2**	**0.4**	**0.6**	**0.8**
3	1 to 10	31 to 40	61 to 70	91 to 100
6	11 to 20	41 to 50	71 to 80	101 to 110
9	21 to 30	51 to 60	81 to 90	111 to 120

5.2.2. Performance Indices

Obviously, the results obtained from multi-objective optimizations comprise a set of solutions rather than a single optimal solution. Of the multi-objective optimization problems, Pareto front DTCTP yields a sequence of solutions for the problems with conflicting objectives of time and cost. The obtained sets of results for Pareto oriented DTCTP are mutually incomparable and non-dominated with respect to multiple objectives of the project. Consequently, the definition of quality is complicated within this context and for Pareto front DTCTP it cannot be evaluated using the concept of optimality which is exercised in single objective optimizations. Owing to this fact, numerous performance indices have been proposed for performance evaluation of multi-objective methods which mainly

engage three aspects of the generated solutions as follows (Zitzler, Deb, and Thiele, 2000):

- The number of the located Pareto optimal solutions – measured using cardinality indices;
- The accuracy of the captured Pareto fronts – measured using accuracy indices;
- The diversity (distribution and spread) of the achieved Pareto optimal solutions – measured using diversity indices.

There is also a second classification criterion which groups performance metrics into unary and binary indexes. The first group defines a value by considering a single set of solutions while the second group assigns relative values for two comparable sets. Overall Non-dominated Vector Generation (ONVG) and Error Ratio (ER) are among indices mainly used for measuring cardinality of the Pareto fronts (Knowles and Corne, 2002; Okabe, Jin, and Sendhoff, 2003). Generational Distance (GD) and Inverted Generational Distance (IGD) are among the indexes mainly utilized in evaluation of the accuracy of the solutions (Zitzler et al., 2000, Okabe et al. 2003, Zhang and Li 2010, Riquelme, Von Lucken, and Baran, 2015). Diversity of the Pareto fronts are measured using indices that include Range Variance (RV), Δ', Uniform Distribution (UD), and Chi-Square-Like Deviation (Zitzler et al., 2000; Okabe et al., 2003; Zhang and Li, 2010). Hypervolume (HV), also known as S-metric is able to capture all three aspects of cardinality, accuracy, and diversity of the Pareto solutions (Zitzler and Thiele, 1999; Riquelme et al., 2015). According to Riquelme et al. (2015), HV, GD, Epsilon Indicator (ε), and IGD are the most practiced indexes in the domain of multi-objective optimization. Knowles and Corne (2002) after studying various indexes recommend utilization of R-metrics (Hansen and Jaskiewicz, 1998) and HV in performance evaluations. By comparing a multitude of performance indices, Okabe et al. (2003) argue that a single unary index cannot adequately reflect all the three aspect of the Pareto fronts and that binary indices are more suitable for comparing different sets of solutions. It is also discussed that the abovementioned indices might occasionally deliver

misleading information since they are not free of cons/caveats (Knowles and Corne, 2002). Meanwhile, it is observed that the convergence speed is not emphasized sufficiently in the literature, however, it is one of the most important aspects of an optimization method. For any practical application of a solution method, the effectiveness needs to be accompanied with the efficiency.

Respecting the findings on the metrics mentioned above, a more holistic approach is taken toward performance evaluation of the approaches introduced in this thesis by employing a collection of unary and binary performance indices along with the CPU times. The unary ONVG metric is used to measure cardinality of the obtained frontiers. A binary metric called ND_{pct} is also introduced in this thesis to measure the cardinality of the Pareto fronts. ND_{pct} measures the percentage of the non-dominated solutions achieved by different optimization methods after discarding the dominated solutions from the unified front, UF. In the cases when true Pareto fronts are available, two sorts (unary and binary types) of Average Percent Deviation (APD) values are calculated to assess accuracy of the solutions. The binary index, APD_{bin}, is calculated for the cost figures based only on the completion duration amounts located mutually by all of the different approaches; and the unary APDs are measured for each approach independently. A normalized, HV-based approach called Hyperarea Ratio (HR) (Veldhuizen, 1999) is adopted to measure convergence and diversity of the frontiers. Unary HR is measured by calculating the ratio as $\frac{\mathrm{HV}_{PF}}{\mathrm{HV}_{TP}}$; where HV_{PF} is the Hypervolume of the approximated Pareto front and HV_{TP} is the Hypervolume of the true Pareto front. In the cases when the true Pareto fronts are unavailable, the binary HR is calculated as $\frac{\mathrm{HV}_{PF}}{\mathrm{HV}_{UF}}$; where HV_{UF} is the Hypervolume of the unified front consisting of the individual solutions in all the approximate frontiers, excluding the dominated ones. Hypervolume indicates the partition of the solution space bounded by the non-dominated frontier and a reference point (R) (Zitzler and Thiele, 1998). For each

$Sol \in Pareto$, an area is constructed using a reference point and Sol as the diagonal corner of the area. The reference point for the first area, i.e., R, is defined by increasing each of the maximum objective function values by 0.5%. The maximum objective function values are derived with regard to both PF and TP in case the optimal front is at hand, or it is set by considering UF if the true Pareto is unavailable. Intersection points of the preceding Sol s and R are appointed as the reference points for the subsequent areas. The union of the constructed areas yields the Hypervolume for that frontier. Consequently, the value of HR belongs to $U(0,1)$; where values close to one suggest the obtained frontier is close to the best-known Pareto front (either TP or UF). Higher values for ONVGs, ND_{pct}, and HR are desirable; whereas, lower values for both the unary and binary APDs and CPU time are preferred. All the performance evaluation procedures have been implemented in C# programming language using Microsoft Visual Studio 2013.

The procedure exercised for Hypervolume calculation of the solution sets is elucidated by means of a hypothetical case example. A shown in Table 5.5, a true Pareto, TP, is assumed to comprise three optimal solutions, on the other hand, two approximated Pareto fronts, PF_a and PF_b, are assumed to include three and five solutions, respectively.

Table 5.5 – Hypothetical Pareto fronts.

TP		PF_a		PF_b	
Dur. (day)	**Cost ($)**	**Dur. (day)**	**Cost ($)**	**Dur. (day)**	**Cost ($)**
44	648,000	44	648,000	44	648,000
53	621,000	53	642,000	53	621,000
64	620,000	62	641,000	64	620,000
		64	637,000		
		69	620,000		

As shown in Figure 5.12, the coordinates of the reference point R are calculated with regard to the maximum duration and cost amounts among all the three sets of

TP, PF_a, and PF_b as follows: $R_{Dur.} = 1.05 \times 69 = 72.45$ and $R_{Cost} = 1.005 \times 648,000 = 651,240$. As a side note, $R_{Dur.}$ in this case example is calculated by augmenting the maximum duration by 5% to obtain an improved visualization of the Hypervolumes; though, both R_{Cost} and $R_{Dur.}$ are set by increasing the worst objective functions by 0.5% in the following sections of this thesis.

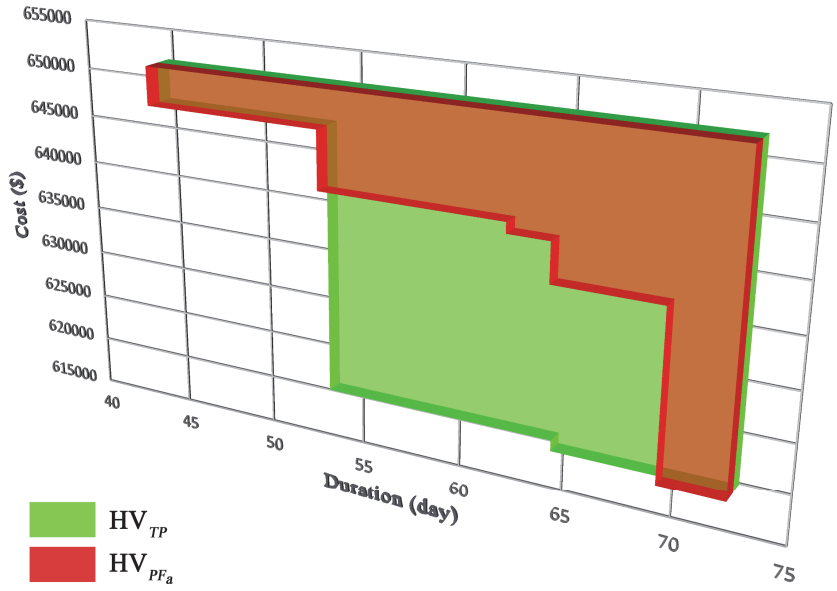


Figure 5.12 – Hypothetical Hypervolume comparison for PF_a.

As demonstrated in Figure 5.12, the area highlighted in red resembles HV_{PF_a} and the area highlighted in green shows HV_{TP}. HR for this set of solutions is calculated as $\mathrm{HR}_a = \frac{163810}{334610} = 0.48$. Similarly, Hypervolume of PF_b is shown in red in Figure 5.13 against the same Hypervolume of TP shown in green. In Figure 5.12 and Figure 5.13 Hypervolumes are extruded to enhance the graphical representation of the areas.

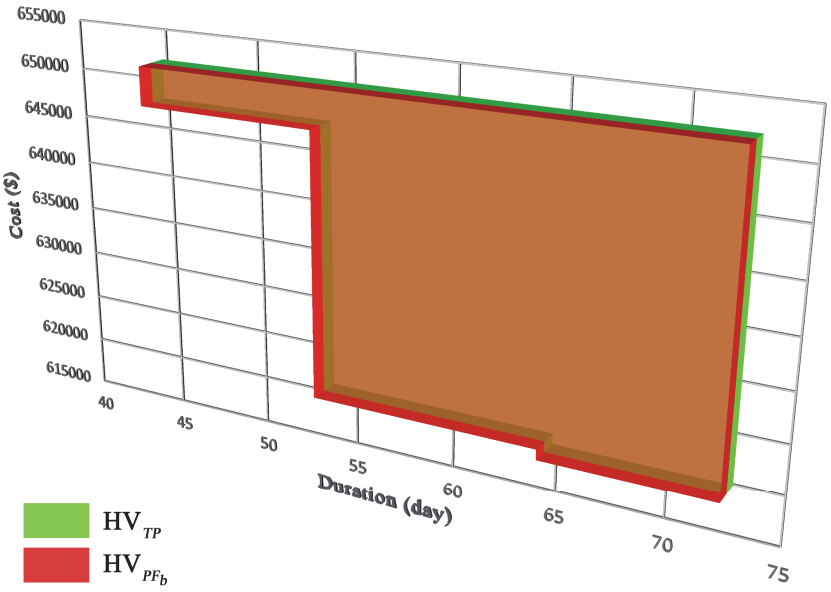


Figure 5.13 – Hypothetical Hypervolume comparison for PF_b.

HR for PF_b is calculated as $\text{HR}_b = \frac{334610}{334610} = 1$. This is a clear indication of why a multitude of performance metrics must be implemented. Although the ONVG of PF_a is greater than that of PF_b, it has been shown that the second set of solutions with $HR = 1$ have an exact compliance with the true Pareto front. PF_b with an HR value of one is shown to outclass PF_a with a lower HR value of 0.48, despite capturing fewer number of non-dominated solutions. Therefore, this thesis study uses the aforementioned performance indexes in a complementary fashion to come up with a more precise verdict on the performances of the proposed approaches. To the best of author's knowledge, Zhang and Li (2010), Kalhor et al. (2011), Fallah-Mehdipour et al. (2012), and Mungle et al. (2013) are among the few studies within the construction management literature reporting on different performance indices of their proposed optimization approaches.

5.2.3. Mixed-Integer Linear Programming Technique

Since all the extensions of DTCTP are Non-deterministic polynomial-time hard (NP-hard) problems in the strong sense (De et al., 1997), only a few studies as discussed in Section 2.3 present exact methods for the DTCTP, especially for the more complex Pareto front problem. The nature of Mixed-Integer Linear Programming model, abridged as MILP (also known as MIP), is well-suited for the solution of DTCTP. The incremental cost-slopes scheme of realistic projects which captures the decline in productivity rate of the crashed activities lead to a convex solution space. The convexity of the solution space guarantees converging to global optimal solutions by means of a MILP algorithm. Any type of time-cost trade-off problem can be modeled using MILP and solved by means of a compatible commercial optimization software. MILP models for DTCTP problems generally include four to five parameters per each activity (Bettemir, 2009); thus, the large-scale problems involve a considerably high number of parameters. Nonetheless, MILP is guaranteed to locate the optima for DTCTP unless there are physical limitations in the memory of the computing devices. Inasmuch as the exact methods are the only approaches guaranteeing optimality of the solutions, they play a crucial role in experimentation of heuristic and meta-heuristic algorithms. In order for adequate evaluation of the effectiveness of the proposed approaches by means of the performance metrics introduced in Section 5.2.2, a MILP model is developed using Gurobi solver version 6.0.5.

Primarily, the proposed MILP employs the serial/parallel merging technique presented in Section 5.1.1.1 and Section 5.1.1.2 to reduce the network of the problem to a simpler equivalent network. As mentioned in Section 5.1.1.1, MILP takes a serial merging approach similar to that of CS-Heuristic described in Section 5.1.3 and Section 5.1.4; however, for the MILP in case of a tie, all but one of the duplicate alternatives with $d_{k_{j1'}} = d_{(k+1)_{j1'}}$ and $dc_{k_{j1'}} = dc_{(k+1)_{j1'}}$ are eliminated. A dominance rule is also implemented to eliminate the dominated components as

$k_{j1'} \succ (k+1)_{j1'}$, which indicates if $d_{k_{j1'}} = d_{(k+1)_{j1'}}$ while $dc_{k_{j1'}} < dc_{(k+1)_{j1'}}$, $(k+1)$th alternative of the equivalent activity j_1' will be eliminated. This procedure is explained using the case example presented in Section 3.3.1. It is shown in Figure 5.1 that the equivalent Activity-1 would have included seven time-cost alternatives in case CS-Heuristic's merging technique was applied. However, for MILP, the dominated time-cost alternative with a duration of 46 days and direct cost of \$59,000 – resulted from adding the third mode of the original Activity-1 to the second mode of the original Activity-3 – is retained and added to the available options (Figure 5.14). The reason behind not discarding the dominated component is simply to prevent generation of incomplete true Pareto fronts. The solution space of the original network comprises 324 realizations; however, the solution space of the reduced network includes 288 different realizations. As is clear, the size of the solutions space shrinks only marginally which might provide a rather small benefit and might not justify the extra efforts involved in merging the activities serially. Owing to this very reason, only the parallel merging technique is exercised for optimal solution of the practiced instances in Section 5.2.5.

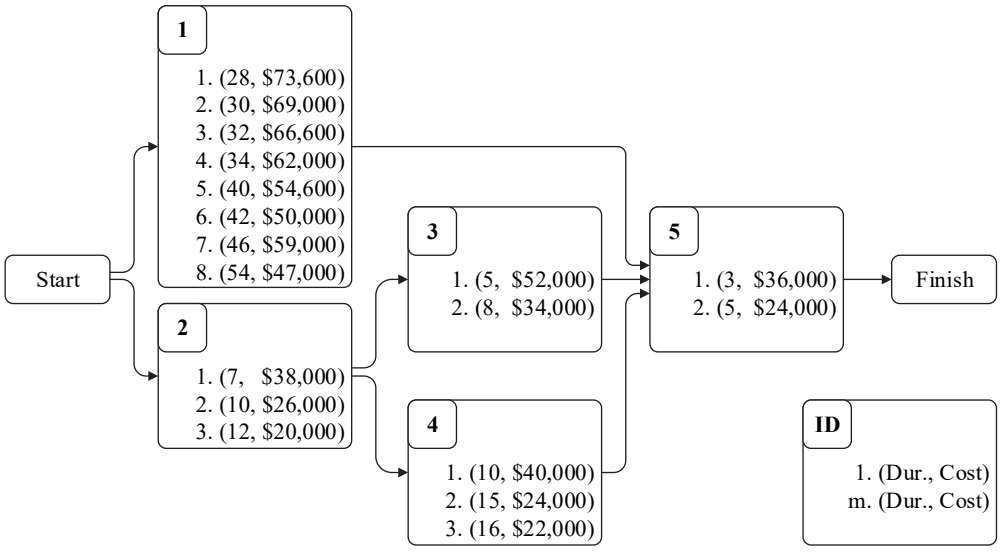


Figure 5.14 – Serial merge applied to the Case Example.

After reducing the problem network, MILP calculates the completion deadline based on the procedure described in Section 5.1.3. The proposed MILP model is based on the modified formulation that explicitly includes bonus (db), delay penalty (dp), and daily indirect cost (ic) rates. The modified formulation for MILP (Bilir, 2015) is presented in Eqs. (1.1)-(1.6) in Section 1.1.2. Gurobi version 6.0.5 is adopted among different commercial optimization software for solution of the MILP model for DTCTP. Gurobi is a high-speed powerful solver with an inclusive library for different programming environments such as C# and C++. Gurobi provides a flexible licensing along with a major collection of mathematical programming solvers, such as MILP. On the other hand, Gurobi is shown to be the fastest solver (Mittelmann, 2013) among different optimization software (e.g., CPLEX, MATLAB) to capture optimality, feasibility, and infeasibility of a set of benchmark problems (MIPLIB2010) which can be found in the "MILP Benchmark" section of the site maintained by Mittelmann.

Typically, Gurobi is designed to read problems in LP-file format. However, an online converter (written as a block of code) is implemented to enable the MILP method to read from the generic txt file format which is universally used in all of the approaches proposed in this thesis. Presolve process, which is a preprocessing technique frequently applied within the MILP-models, is experienced to contribute to a larger computational time. Therefore, it is excluded from the optimization model. Two MILP models are designed and developed for deadline and Pareto front DTCTP separately. For Pareto front problem, upper and lower boundaries are set for the duration of the problem in accord with all-normal, $CPM_{\max}$, and all-crashed, $CPM_{\min}$, schedules, respectively. Moreover, a horizon-varying approach is also implemented for unraveling Pareto front problem which involves iterative solution of deadline DTCTP by varying D_{dl} from $CPM_{\min}$ to $CPM_{\max}$, in steps of one. Pareto oriented MILP is also complemented with an upper-bound calculation mechanism for the cost of the feasible durations. The value of the upper-bound is designed to update to the cost figure located in the preceding iteration of the

horizon-varying approach. This is facilitated by means of the Cutoff parameter of Gurobi which basically indicates that the solver should only consider solutions with total costs less than the specified amount. Inclusion of the upper-bound rules out the need for elimination of the dominated solutions from the final Pareto, since, it only locates the non-dominated solutions in the course of the optimization.

Simulation routines of the proposed MILP method are coded in C# and compiled within Visual Studio 2013 on a 64-bit platform. All of the tests are carried out on a desktop computer with a P9X79 Chipset motherboard, 16 GB 667 MHz DDR3 random-access memory (RAM), Intel Core i7-3.40 GHz CPU, and 64-bit Windows 10 operating system. MILP is executed solely (no other software is ran simultaneously) on a single processor and overclocking is not performed. All the runs are truncated after a CPU time of one hour which is enforced using TimeLimit parameter of Gurobi. If it exceeds the runtime limit, the algorithm will terminate by reporting a non-optimal status.

The new sets of instances presented in Section 5.2.1 are fitted into the proposed MILP model in order to locate the optimal solutions for deadline and Pareto front DTCTPs. The percentage of the instances solved within the enforced computation time limit of one hour are demonstrated for deadline and Pareto front DTCTPs in Table 5.6 and Table 5.8, respectively. These tables summarize percentages for every combination of different network sizes, network complexities, and mode numbers. The average CPU times required by MILP method for tackling every ten replicate problems of each parameter level combination are illustrated in Table 5.7 and Table 5.9 for deadline and Pareto front DTCTPs, respectively. Unavailable values are tabulated as ‘na’ in Table 5.7 and Table 5.9.

Table 5.6 – Percentage of RanGen2 instances optimally solved for deadline DTCTP.

% of Solved Instances																				
# of Acts.	**50**				**100**				**200**				**500**				**990**			
I_2	**0.2**	**0.4**	**0.6**	**0.8**	**0.2**	**0.4**	**0.6**	**0.8**	**0.2**	**0.4**	**0.6**	**0.8**	**0.2**	**0.4**	**0.6**	**0.8**	**0.2**	**0.4**	**0.6**	**0.8**
# of Modes																				
3	100	100	100	100	100	100	100	100	100	50	100	100	0	100	100	100	0	0	0	100
6	100	100	100	100	100	100	100	100	70	0	100	100	0	20	20	100	0	0	0	100
9	100	100	100	100	90	100	100	100	20	20	100	100	0	30	20	100	0	0	0	100

Table 5.7 – Average CPU time of MILP for RanGen2 instances solved for deadline DTCTP.

Avg. CPU time (s)																				
# of Acts.	**50**				**100**				**200**				**500**				**990**			
I_2	**0.2**	**0.4**	**0.6**	**0.8**	**0.2**	**0.4**	**0.6**	**0.8**	**0.2**	**0.4**	**0.6**	**0.8**	**0.2**	**0.4**	**0.6**	**0.8**	**0.2**	**0.4**	**0.6**	**0.8**
# of Modes																				
3	0.14	0.07	0.06	0.03	4.81	0.73	0.17	0.07	461.92	1333.29	0.61	0.12	na	46.33	13.14	0.44	na	na	na	1.50
6	1.13	0.28	0.24	0.08	243.77	8.29	1.46	0.15	718.51	na	4.01	0.25	na	615.82	1407.16	3.08	na	na	na	12.24
9	1.27	0.29	0.43	0.11	177.43	20.89	3.39	0.21	171.36	1440.83	10.19	0.40	na	1149.87	598.14	5.22	na	na	na	27.10

Table 5.8 – Percentage of RanGen2 instances optimally solved for Pareto front DTCTP.

% of Solved Instances																				
# of Acts.	**50**				**100**				**200**				**500**				**990**			
I_2	**0.2**	**0.4**	**0.6**	**0.8**	**0.2**	**0.4**	**0.6**	**0.8**	**0.2**	**0.4**	**0.6**	**0.8**	**0.2**	**0.4**	**0.6**	**0.8**	**0.2**	**0.4**	**0.6**	**0.8**
# of Modes																				
3	100	100	100	100	0	90	100	100	0	0	100	100	0	0	0	100	0	0	0	100
6	100	100	100	100	0	0	70	100	0	0	30	100	0	0	0	80	0	0	0	0
9	50	100	100	100	0	0	60	100	0	0	0	100	0	0	0	0	0	0	0	0

Table 5.9 – Average CPU time of MILP for RanGen2 instances solved for Pareto front DTCTP.

Avg. CPU time (s)																				
# of Acts.	**50**				**100**				**200**				**500**				**990**			
I_2	**0.2**	**0.4**	**0.6**	**0.8**	**0.2**	**0.4**	**0.6**	**0.8**	**0.2**	**0.4**	**0.6**	**0.8**	**0.2**	**0.4**	**0.6**	**0.8**	**0.2**	**0.4**	**0.6**	**0.8**
# of Modes																				
3	50.27	18.43	14.28	11.26	na	1516.12	91.81	40.55	na	na	355.33	118.01	na	na	na	664.24	na	na	na	2377.88
6	1210.03	158.37	79.80	26.04	na	na	949.52	89.37	na	na	2175.79	246.95	na	na	na	2376.24	na	na	na	na
9	2468.87	293.68	188.24	37.96	na	na	1453.82	125.76	na	na	na	359.68	na	na	na	na	na	na	na	na

The contents of Table 5.6 and Table 5.7 reveal MILPs successful convergence to optimum solutions for deadline problems that include up to 100 activities by solving an overall 99.16% of 50-activity and 100-activity instances within the imposed runtime limit. It is displayed that on average, MILP requires 0.35 and 38.45 seconds to locate the optimum solutions for 50-activity and 100-activity problems, respectively. The performance of MILP is consistent for the larger network of size 200 and 500. MILP is shown to be able to solve all of the 200-activity and 500-activity problems with I_2 values of 0.6 and 0.8 which contain virtually serial networks. Overall, 43.33% of 200-activity and 25% of 500-activity problems with smaller I_2 values are solved to optimality which indicates the networks with almost parallel graphs require greater processing times. MILP is experimented to locate the global optimum solutions in 376.50 and 426.58 seconds on average, for 200-activity and 500-activity instances, respectively. MILP is also shown to be able to unravel 25% of the instances with 990 activities, comprising 3 to 9 time-cost alternatives. CPU time limit of one hour is experienced to be insufficient for the more complex (with regard to the nature of MILP) 990-activity instances with I_2 values of 0.2 to 0.6; however, MILP is displayed to be capable of solving 3 sets of 990-activity problems, i.e., 30 instances, with I_2 values of 0.8 in an average processing time of 13.61 seconds.

The contents of Table 5.8 and Table 5.9 shed some light on the performance of MILP over capturing the true Pareto front of the complex RanGen2 instances. It is revealed that MILP is virtually able to locate the true Pareto fronts for all the 50-activity instances in an average processing time of 379.77 seconds. Solving 50% of the most complex (with regard to the nature of MILP) set of 50-activity problems with I_2 values of 0.2 and 9 time-cost modes, MILP obtains exact solutions for an overall 95.83% of the 50-activity instances. MILP is shown to be able to solve all the pseudo-serial 100-activity and 200-activity problems with I_2 values of 0.8. MILP is displayed to be able to capture optimal frontiers for 53.33% of the 100-

activity problems with smaller I_2 values of 0.4 and 0.6. With an overall average computation time of 609.56 seconds, MILP is experimented to require more than one hour to solve 100-activity problems with I_2 values of 0.2. 43.33% of the 200-activity problems with I_2 value of 0.6 are solved for Pareto front problem within the enforced time limit. With an overall average processing time of 651.15 seconds, runtime limit of one hour is experienced to be inadequate for capturing the true Pareto fronts for pseudo-parallel 200-activity problems with smaller I_2 values of 0.2 and 0.4. The successful performance of MILP is consistent for the larger sets of instances with 500 and 990 activities that include 3 modes with network of $I_2 = 0.8$. MILP requires an overall average computation time of 1520.24 seconds to locate the efficient frontiers for 15% of the 500-activity problems. An overall average of 2377.38 seconds is recognized to be sufficient for solution of 8.3% of the largest set of instances with 990 activities.

In the light of the above interpretations, an overall picture of MILP's performance can be obtained as the nature of this method tends to be more suitable for pseudo-serial networks with greater I_2 values that include smaller number of time-cost alternatives. Thus, problems 21 to 30 are experimented to be the most complicated, and the problems 91 to 100 to be the simplest problems for the proposed exact algorithm.

In the absence of the upper-bound (Cutoff parameter) and parallel merging (Section 5.1.1.2) techniques, the proposed MILP is able to obtain optimal Pareto fronts for 207 number of instances in an overall average processing time of 656.06 seconds; however, implementation these techniques increases the number of solved problems by more than 19% while reducing the overall average CPU time by more than 18%. To the best of author's knowledge, this is the first contribution where global optimal costs and true Pareto fronts are captured for real-life-scale instances that are based upon the complex RanGen2 networks. With regard to the summary of the past research given in Table 2.1, it can also be observed that no previous

exact method, other than Bilir's (2015) approach, has successfully been applied to large-scale DTCT problems. Nevertheless, MILP is proved to be capable of locating the optima for instances with up to 990 activities.

5.2.4. Cost Minimization and Deadline DTCTPs

Computational experiments are carried out to evaluate the performance of the proposed CS-Heuristic method for cost minimization and deadline DTCTPs using a set of benchmark instances acquired from the literature, based on the performance metrics presented in Section 5.2.2. The proposed optimization algorithm is coded in C# and compiled within Visual Studio 2013 on a 64-bit platform. All of the tests are carried out on a desktop computer with a P9X79 Chipset motherboard, 16 GB 667 MHz DDR3 random-access memory (RAM), Intel Core i7-3.40 GHz CPU, and 64-bit Windows 10 operating system. CS-Heuristic is executed solely (no other software is ran simultaneously) on a single processor and overclocking is not performed.

5.2.4.1. Small-Scale Benchmark Problems

The performance of the proposed CS-Heuristic for deadline DTCTP is first tested using the small-scale benchmark instances which are commonly used in the literature (Elbeltagi et al., 2005; Zheng et al., 2005; El-Rayes and Kandil, 2005; Kandil and El-Rayes, 2006; Elbeltagi et al., 2007; Ng and Zhang, 2008; Xiong and Kuang, 2008; Afshar et al., 2009; Fallah-Mehdipour et al., 2012; Sonmez and Bettemir, 2012; Zhang and Ng, 2012; Monghasemi at al., 2015) for performance evaluations. Readers are referred to Section 3.4.2 for details on the practiced small-scale instances including 18a, 18b, and 18c problems. Performance of CS-Heuristic is compared with DPSO algorithm (Section 3.3) which is shown in Section 3.4.2 to be able to outperform any of the previous optimization methods. Snapshots of the

performance of the results for problems 18a, 18b, and 18c are given in Table 5.10, Table 5.11, and Table 5.12, respectively.

Table 5.10 – Performance of CS-Heuristic for problem 18a.

Algorithm	CPU Time (s)	APD (%)
DPSO (Section 3.3)	0.4	0.00
CS-Heuristic (Section 5.1.3)	~0	0.00

Table 5.11 – Performance of CS-Heuristic for problem 18b.

Algorithm	CPU Time (s)	APD (%)
DPSO (Section 3.3)	0.4	0.00
CS-Heuristic (Section 5.1.3)	~0	0.00

Table 5.12 – Performance of CS-Heuristic for problem 18c.

Algorithm	CPU Time (s)	APD (%)
DPSO (Section 3.3)	0.4	0.00
CS-Heuristic (Section 5.1.3)	~0	0.00

According to the contents of the above tables, it is observed that CS-Heuristic is also able to locate the global optimum solutions for problems 18a, 18b, and 18c; however, by running on the same desktop computer, the processing time of this method is revealed to be less than that of the DPSO algorithm for all the small-scale instances.

5.2.4.2. Medium-Scale Benchmark Problems

The performance of the proposed CS-Heuristic for deadline DTCTP is also tested using the medium-scale problem of Sonmez and Bettemir (2012). Readers are referred to Section 3.4.3 for the explanations on the exercised medium-scale problems including 63a and 63b problems. Results of CS-Heuristic are compared with the DPSO algorithm (Section 3.3), detailed results of which was demonstrated in Section 3.4.3. DPSO was conceded to be more effective and efficient than both

the GA and HA methods of Sonmez and Bettemir (2012). Results of CS-Heuristic for the medium-scale instances are tabulated in Table 5.13. Unavailable values are tabulated as 'na' in Table 5.13.

Table 5.13 – Performance of CS-Heuristic for problems 63a and 63b.

Algorithm	63a		63b	
	CPU Time (s)	APD (%)	CPU Time (s)	APD (%)
NAA (Bettemir and Birgonul, 2017)	na	0.04	na	0.07
Parallel GA (Agdas et al., 2018)	111	0.26	88.8	1.24
DPSO (Section 3.3)	1.3	0.02	1.3	0.05
CS-Heuristic (Section 5.1.3)	0.01	0.08	0.01	0.07

The comparison of CS-Heuristic with the state-of-the-art methods proves that the proposed CS-Heuristic is among the top performing algorithms for the medium-scale deadline DTCTP. CS-Heuristic is displayed to outperform DPSO – using the same PC – and Parallel GA of Agdas et al. (2018) with regard to the required processing time. Bettemir and Birgonul (2017) do not report the computation time requirement of their Network Analysis Algorithm (NAA); though, its convergence capabilities are observed to be virtually equal to CS-Heuristic. Besides, accuracy of the solutions obtained by CS-Heuristic while being on par with DPSO, is shown to be significantly better than Parallel GA of Agdas et al. (2018).

5.2.4.3. Large-Scale Benchmark Problems

The large-scale problems used for performance measurement of the proposed CS-Heuristic include problems comprising 630, 1800, 3150, and 6300 activities. The 630-activity, 3150-activity, and 6300-activity problems are generated by copying the 63-activity problem of Sonmez and Bettemir (2012) ten, 50, and 100 times in serial, respectively. Readers are referred to Section 3.4.4 for the description of the 63-activity-based large-scale problems. The 1800-activity problem, on the other hand, is generated by copying the 18-activity problem of Hegazy (1999) 100 times

in serial. Readers are referred to Section 4.4.2 for details on the base 18-activity problem.

The performance of CS-Heuristic is tested against DPSO (Section 3.3) and Parallel GA of Agdas et al. (2018). In Section 3.4.4, it is illustrated that DPSO is significantly faster and more effective that GA and HA approaches of Sonmez and Bettemir (2012). Performance of the results for 630-activity problems are summarized in Table 5.14.

Table 5.14 – Performance of CS-Heuristic for problems 630a and 630b.

Algorithm	630a		630b	
	CPU Time (s)	APD (%)	CPU Time (s)	APD (%)
Parallel GA (Agdas et al., 2018)	364.2	2.76	396.6	2.29
DPSO (Section 3.3)	14.6	0.33	14.6	0.34
CS-Heuristic (Section 5.1.3)	5.23	0.05	5.15	0.10

Agdas et al. (2018) experimented 630-activity problems by applying parallel computing on a high-performance computing facility that included eight CPU cores. Though, both DPSO and CS-Heuristic running on a single CPU are shown to remarkably outperform Parallel GA of Agdas et al. (2018) with substantially smaller deviation and runtime amounts. CS-Heuristic is also observed to best DPSO with respect to both accuracy and the convergence speed.

The performance of CS-Heuristic is also investigated using the 1800-activity problems which are used by Agdas et al. (2018). This problem is examined under two different conditions. In problem 1800a, the indirect cost rate is $200/day, the delay penalty is set as $20,000/day, and the incentive payment is assumed as $1,000/day. Whereas, in problem 1800b, the indirect cost rate is assumed to be $1,500/day. CS-Heuristic is experimented against the Parallel GA of Agdas et al. (2018), results of which are presented in Table 5.15.

Table 5.15 – Performance of CS-Heuristic for problems 1800a and 1800b.

Algorithm	1800a		1800b	
	CPU Time (s)	APD (%)	CPU Time (s)	APD (%)
Parallel GA (Agdas et al., 2018)	20,952	7.05	21,024	14.72
CS-Heuristic (Section 5.1.3)	40.91	0.00	40.66	0.00

CS-Heuristic is experienced to be able to locate the global optimum solutions for both the 1800a and 1800b problems within a processing time of less than 41 seconds. Performance of CS-Heuristic is incomparable with Parallel GA (Agdas et al., 2018) for this large-scale problem, since, Parallel GA can find solutions with considerably larger deviations in substantially longer computation times of more than five hours.

The CS-Heuristic is further experimented using the 3150-activity problems of Agdas et al. (2018). This problem is examined under two different conditions. Indirect cost rate for problems 3150a and 3150b are set as \$2,300/day and \$3,500/day, respectively. Performance of CS-Heuristic is compared with Parallel GA of Agdas et al. (2018) and the results are given in Table 5.16.

Table 5.16 – Performance of CS-Heuristic for problems 3150a and 3150b.

Algorithm	3150a		3150b	
	CPU Time (s)	APD (%)	CPU Time (s)	APD (%)
Parallel GA (Agdas et al., 2018)	32,940	6.5	33,876	4.73
CS-Heuristic (Section 5.1.3)	548.36	0.04	529.43	0.10

It is discovered that the proposed CS-Heuristic can converge to global optimum solutions with only fractional deviations for 3150-activity problems. For this problem, as well, CS-Heuristic is shown to greatly outperform Parallel GA of Agdas et al. (2018) on the grounds of efficiency and effectiveness. Computational time of Parallel GA running on eight cores of a high-performance computing facility is measured to be nine hours more than that of CS-Heuristic.

Finally, performance of the proposed CS-Heuristic evaluated using the 6300-activity problems of Agdas et al. (2018). This problem is also implemented under two different settings. Indirect cost rate for problems 6300a and 6300b are assigned as $2,300/day and $3,500/day, respectively. As shown in Table 5.17, performance of CS-Heuristic is measured by directing comparisons with Parallel GA of Agdas et al. (2018).

Table 5.17 – Performance of CS-Heuristic for problems 6300a and 6300b.

Algorithm	6300a		6300b	
	CPU Time (s)	APD (%)	CPU Time (s)	APD (%)
Parallel GA (Agdas et al., 2018)	59,112	7.66	60,336	6.96
CS-Heuristic (Section 5.1.3)	4484.56	0.04	4292.81	0.10

The successful performance of CS-Heuristic is also consistent for the 6300-activity problems. It is proved that the proposed CS-Heuristic is capable of converging to global optimal solutions with only fractional deviations for problems 6300a and 6300b. Performance of CS-Heuristic is unmatched by Parallel GA (Agdas et al., 2018) for the largest and the most complex problem implemented. There is remarkable gap between the performance of the proposed CS-Heuristic and the Parallel GA for solution of this instance. Processing time requirement of Parallel GA running on eight cores of a high-performance computing facility is revealed to be more than 15 hours over the CPU time requirement of the proposed CS-Heuristic. The average percent deviation of Parallel GA is measured to be almost 191 and 70 times the amount of APDs for CS-Heuristic over problems 6300a and 6300b, respectively.

Comparative studies reveal superiority of the proposed CS-Heuristic over earlier approaches as well as the DPSO method. It is obvious that not only the computation time requirement of the innovative optimization model presented in this section is remarkably less than the earlier approaches, but also it is able to locate high quality solutions for all the practiced cost minimization/deadline problems. Owing to its

unprecedented efficacy and exceptional accuracy, Cost-Slope Heuristic is expected to contribute to optimal planning of real-life-scale construction projects. To the best of author's knowledge, the proposed CS-Heuristic optimization model is the first method that outperforms state-of-the art meta-heuristics and is capable of unraveling large-scale problems comprising thousands of activities within practically reasonable timeframes with only fractional deviations.

5.2.5. Pareto front DTCTP

Computational experiments are carried out to measure the performance of the proposed CS-Heuristic method for Pareto front DTCTPs using a set of benchmark and case problems acquired from the literature as well as the RanGen2 instances introduced in Section 5.2.1, based on the performance indices presented in Section 5.2.2. The proposed optimization algorithm is coded in C# and compiled within Visual Studio 2013 on a 64-bit platform. All of the tests are carried out on a desktop computer with a P9X79 Chipset motherboard, 16 GB 667 MHz DDR3 random-access memory (RAM), Intel Core i7-3.40 GHz CPU, and 64-bit Windows 10 operating system. CS-Heuristic is executed solely (no other software is ran simultaneously) on a single processor and overclocking is not performed.

5.2.5.1. Small-Scale Benchmark Problems

The performance of the proposed CS-Heuristic for Pareto front DTCTP is first tested using the small-scale benchmark instances that include 18 activities and up to five time-cost modes. This problem is widely used in the literature (Elbeltagi et al., 2005; Zheng et al., 2005; El-Rayes and Kandil, 2005; Kandil and El-Rayes, 2006; Elbeltagi et al., 2007; Ng and Zhang, 2008; Xiong and Kuang, 2008; Afshar et al., 2009; Fallah-Mehdipour et al., 2012; Sonmez and Bettemir, 2012; Zhang and Ng, 2012; Monghasemi at al., 2015) for performance evaluations. Readers are referred to Section 4.4.2 for details on the practiced small-scale instances including

18d, 18e, 18f, and 18g problems. Performance of CS-Heuristic is compared with PFPSO model (Section 4.3) which is shown in Section 4.4.2 to be able to surpass any of the earlier optimization methods in the literature. Results of the performance assessment over problems 18d, 18e, 18f, and 18g are displayed in Table 5.18.

Table 5.18 – Performance comparison of CS-Heuristic for small-scale problems.

Method	Problem	ONVG	ND_{pct} (%)	APD (%)	APD_{bin} (%)	HR	CPU Time (s)
PFPSO (Section 4.3)	18d	39	100	0	0	1	2
	18e	44	100	0	0	1	2
	18f	18	100	0	0	1	2
	18g	4	100	0	0	1	2
CS-Heuristic (Section 5.1.4) with PM*	18d	39	87.18	0.01	0.01	1	0.03
	18e	43	84.09	0.01	0.01	0.99	~0
	18g	15	83.33	0	0	1	~0
	18d	4	100	0	0	1	~0
CS-Heuristic (Section 5.1.4) with SM**	18d	39	87.18	0.01	0.01	1	0.01
	18e	42	75	0.02	0.02	0.99	0.01
	18f	10	55.56	0	0	0.99	~0
	18g	4	100	0	0	1	~0

*Parallel Merge

**Serial Merge

Performance of the proposed CS-Heuristic is experimented by using either of the merging techniques of serial and parallel, independently. Compared with PFPSO, both variants of CS-Heuristic are found to be able to locate the same number of non-dominated solutions for problems 18d and 18g. For the rest of the small-scale problems, the number of the obtained Pareto solutions by PFPSO are slightly more than CS-Heuristic with parallel merge which is followed by CS-Heuristic with serial merge. For problems 18d, 18e, and 18f, PFPSO is observed to perform marginally better than the CS-Heuristics with regard to the performance indexes, excluding the computation time. Results of all the methods are identical for problem 18g with respect to any of the metrics. Nonetheless, executed on the same desktop computer, CS-Heuristics are experienced to require considerably less processing

time for all the 18d, 18e, 18f, and 18g problems which might justify the slight differences in the remainder of the performance indices. CS-Heuristic with serial merge, on the other hand, is shown to perform very close to CS-Heuristic with parallel merge; though, according to ND_{pct} values, parallel merge variant is capable of capturing solutions of higher quality for problems 18e and 18f.

5.2.5.2. Medium-Scale Benchmark Problem

The performance of the proposed CS-Heuristic for Pareto front DTCTP is also evaluated using the medium-scale problem of Kandil and El-Rayes (2006). Readers are referred to Section 4.4.3 for the explanations on the exercised medium-scale 180-activity problem. Results of both the variants of CS-Heuristic are compared with PFPSO method (Section 4.3) detailed results of which was illustrated in Section 4.4.3. PFPSO was validated to be more effective and efficient than both the GP-GA and CG-GA approaches of Kandil and El-Rayes (2006). Results of CS-Heuristics for the medium-scale instance are tabulated in Table 5.19.

Table 5.19 – Comparison of the results for 180-activity problem.

Algorithm	**ONVG**	ND_{pct} **(%)**	**APD (%)**	APD_{bin} **(%)**	**HR**	**CPU Time (s)**
PFPSO (Section 4.3)	304	34.86	0.09	0.09	0.99	21
CS-Heuristic (Section 5.1.4) with PM*	586	69.95	0.27	0.21	0.99	1.37
CS-Heuristic (Section 5.1.4) with SM**	586	69.95	0.27	0.21	0.99	1.25

*Parallel Merge
**Serial Merge

Both variants of the proposed CS-Heuristic approach are able to capture 586 non-dominated solutions in slightly over one second with average deviations of less than 0.3%. CS-Heuristics are able to position a rather larger set of non-dominated solutions along the efficient frontier compared with PFPSO. All the approached are observed to achieve solutions with comparable diversities along the frontiers since

they share the same values for HR index. Performed on the same PC, the processing time requirements of CS-Heuristics are validated to be remarkably less than 21 seconds of PFPSO approach. Although both the unary and binary average deviation amount of PFPSO are smaller than CS-Heuristic, it is indicated by ND_{pct} metric that the number of non-dominated solutions in the final unified front is significantly higher for CS-Heuristic. Faster convergence speed, larger ONVG, and larger ND_{pct} values of CS-Heuristic are believed to make up for its slightly larger deviation amounts. Computational time of serial variant of CS-Heuristic is experimented to be marginally shorter than the parallel variant for this instance.

5.2.5.3. Large-Scale Benchmark Problems

Experiments on large-scale benchmark instances comprise the 360-activity and 720-activity problems created by Kandil and El-Rayes (2006). Readers are referred to Section 4.4.4 for the description of the 18-activity-based large-scale problems. Efficacy and efficiency of the two variants of CS-Heuristic are validated in comparison with PFPSO method (Section 4.3). PFPSO was displayed in Section 4.4.4 to substantially outperform both the GP-GA and CG-GA approaches of Kandil and El-Rayes (2006). Performance of the results for 360-activity problem is summarized in Table 5.20.

Table 5.20 – Comparison of the results for 360-activity problem.

Algorithm	**ONVG**	ND_{pct} **(%)**	**HR**	**CPU Time (s)**
PFPSO (Section 4.3)	536	32.48	0.99	43
CS-Heuristic (Section 5.1.4) with Parallel Merge	1176	71.71	0.99	11.26
CS-Heuristic (Section 5.1.4) with Serial Merge	1176	71.71	0.99	10.04

Unary and binary APDs are not reported for this problem since the true Pareto fronts were not available. Both the CS-Heuristics with parallel and serial merge techniques are able to obtain 1176 non-dominated solutions in slightly over 10 and

11 seconds respectively. CS-Heuristics are able to capture more than two times the amount of non-dominated solutions achieved by PFPSO. All the optimization methods are revealed to be able to locate non-dominated solutions with commensurate diversities along the fronts since they share the same values for HR metric. Performed on the same desktop computer, the processing time requirements of CS-Heuristics are experienced to be almost one fourth of the 43 seconds required by PFPSO method. With respect to the values calculated for ND_{pct} index, the number of non-dominated solutions in the final unified front for CS-Heuristic is more than two times the amount of solutions positioned by PFPSO. CS-Heuristic with serial merge, on the other hand, is observed to require slightly less processing time for solution of this problem in comparison with the parallel variant.

The CS-Heuristic is further experimented using the large-scale 720-activity problems of Kandil and El-Rayes (2006). Performance of CS-Heuristics are compared with PFPSO (Section 4.3) and the results are given in Table 5.21. PFPSO's exceptional convergence capabilities was proved in Section 4.4.4 which outperformed GP-GA and CG-GA approaches of Kandil and El-Rayes (2006) by a large margin.

Table 5.21 – Comparison of the results for 720-activity problem.

Algorithm	**ONVG**	ND_{pct} **(%)**	**HR**	**CPU Time (s)**
PFPSO (Section 4.3)	1022	27.74	0.99	92
CS-Heuristic (Section 5.1.4) with Parallel Merge	2356	75.36	0.99	108.29
CS-Heuristic (Section 5.1.4) with Serial Merge	2356	75.36	0.99	95.47

Unary and binary APDs are not reported since the true Pareto fronts were not known for this problem. The successful performances of CS-Heuristics are also consistent for the large-scale 720-activity problems. It is displayed that the proposed CS-Heuristics are capable of capturing 2356 non-dominated solutions along the efficient frontier within less than two minutes. Both the variants of CS-Heuristic

are able to achieve more than two times the amount of solutions located by PFPSO. While computational time requirement of CS-Heuristic with serial merge is virtually the same as PFPSO, the parallel variant is shown to demand moderately larger processing time on the same desktop computer. Indicated by equal HR values, all the optimization methods are capable of locating non-dominated solutions with comparable distribution and spread characteristics along the frontier. In accord with the values calculated for ND_{pct} index, the number of non-dominated solutions in the final unified front for CS-Heuristic is slightly shy of three time the amount of solutions obtained by PFPSO.

Comparative studies reveal that the performance of CS-Heuristic is unmatched by any of the previous approaches including the PFPSO algorithm. Not only the computation time requirement of the innovative multi-objective optimization model presented in this section is substantially less than the earlier approaches, but it is also able to produce a large number of high quality non-dominated solutions for all the practiced Pareto front problems. Owing to its unprecedented efficacy and exceptional accuracy, Cost-Slope Heuristic is expected to contribute to optimal planning of realistic construction projects. To the best of author's knowledge, the proposed CS-Heuristic optimization model is the first method capable of tackling large-scale problems consisting of hundreds of activities within reasonably short timespans and practically viable deviations.

5.2.5.4. New Sets of Instances

In this section, the new sets of realistic instances are used to compare the performance of PFPSO (Section 4.3) with CS-Heuristic (Section 5.1.4) on the basis of the performance comparison metrics presented in Section 5.2.2. Initially, all the instances are fitted into CS-Heuristic with parallel merge; afterwards, processing time required for solution of each of the 600 instances are extracted to be used as a runtime limit for PFPSO method. Hence, PFPSO is slightly modified to

accommodate new termination criteria. It is designed to terminate either if the enforced runtime limit or the maximum number of iterations is reached. The values for various performance indices for PFPSO and the two variants of CS-Heuristic, i.e., serial and parallel, over the new complex instances are reported in the following tables as follows. Table 5.22 and Table 5.23 compare cardinality of the solutions using ONVG and ND_{pct} values, respectively. Table 5.24 and Table 5.25 shed light on the accuracy of the results based on APD and APD_{bin} values, respectively. Table 5.26 illustrates the convergence and diversity of the Pareto fronts using HR metric. Finally, Table 5.27 displays the convergence speed of each of the approaches by running on the same desktop computer. These tables summarize values for every combination of different network sizes, network complexities, and mode numbers. The average values for every ten replicate problems per each parameter setting are elucidated for the proposed optimization approaches. Unavailable values are tabulated as 'na' in Table 5.24 and Table 5.25.

Table 5.22 – Comparison of ONVG values.

		ONVG																			
	# of Acts.	**50**				**100**				**200**				**500**				**990**			
	I_2	**0.2**	**0.4**	**0.6**	**0.8**	**0.2**	**0.4**	**0.6**	**0.8**	**0.2**	**0.4**	**0.6**	**0.8**	**0.2**	**0.4**	**0.6**	**0.8**	**0.2**	**0.4**	**0.6**	**0.8**
	# of Modes																				
PFPSO	**3**	45	23	11	4	80	43	18	7	145	91	32	7	393	173	70	15	699	400	131	16
	6	96	60	28	8	181	119	42	15	303	241	93	15	902	377	191	3	1590	1022	248	1
	9	136	87	45	17	254	183	77	19	431	368	139	24	1273	550	308	35	2226	1540	201	1
CS-H-PM	**3**	80	64	40	11	190	159	80	32	313	341	127	30	1128	271	175	63	1496	779	346	126
	6	164	130	77	25	324	285	91	34	555	555	98	28	1695	200	147	82	2166	587	207	127
	9	200	149	93	28	385	330	98	27	618	564	59	26	1900	154	85	70	2330	519	175	144
CS-H-SM	**3**	80	63	43	14	190	158	89	38	313	348	138	46	1127	316	210	96	1497	842	328	137
	6	164	137	85	30	324	292	113	47	555	562	158	50	1689	306	165	78	2150	752	192	73
	9	200	155	96	37	385	345	135	45	618	607	123	45	1902	222	113	51	2409	628	122	71

Table 5.23 – Comparison of ND_{pct} values.

ND_{pct} (%)																				
# of Acts.		**50**				**100**				**200**				**500**				**990**			
	I_2	**0.2**	**0.4**	**0.6**	**0.8**	**0.2**	**0.4**	**0.6**	**0.8**	**0.2**	**0.4**	**0.6**	**0.8**	**0.2**	**0.4**	**0.6**	**0.8**	**0.2**	**0.4**	**0.6**	**0.8**
	# of Modes																				
PFPSO	**3**	0	0.48	2.42	7.73	0	0	0.41	1.86	0	0	0.13	1.38	0	0.10	0	0.37	0	0.86	3.46	0.09
	6	0.12	0.07	1.21	2.63	0	0	0	1.95	0	0	3.20	1.62	0	12.86	29.60	0	0	33.52	41.57	0
	9	0.05	0.39	1.85	3.36	0	0	3.25	4.95	0	0.64	19.40	7.70	0	48.81	62.40	16.47	0.48	54.98	33.79	0
CS-H-PM	**3**	100	94.45	75.60	67.33	100	82.60	48.89	44.08	100	53.02	38.41	39.42	97.69	23.37	21.33	22.64	93.93	37.55	17.75	21.75
	6	99.88	81.47	50.64	34.29	100	64.44	31.89	33.63	100	42.78	17.09	17.48	72.23	12.28	6.73	26.31	77.88	10.75	17.60	45.76
	9	100	67.82	50.89	36.35	100	59.96	21.05	20.47	100	39.18	10.63	13.50	64.31	11.58	4.54	32.57	92.29	7.63	42.58	66.26
CS-H-SM	**3**	100	89.15	76.57	84.14	100	92.54	71.94	79.15	100	72.34	79.73	89.28	98.90	83.75	77.37	83.49	96.90	59.71	75.57	78.30
	6	99.88	92.83	67.69	65.66	100	72.73	72.08	76.44	100	52.23	80.49	88.89	77.71	78.33	52.64	75.65	72.72	47.19	37.48	53.79
	9	100	77.61	74.74	73.62	100	47.41	66.54	87.10	100	59.14	73.09	84.48	71.65	36.66	30.92	53	92.66	29.71	18.19	34.25

Table 5.24 – Comparison of APD values.

APD (%)																				
	# of Acts.	**50**				**100**				**200**				**500**				**990**			
	I_2	**0.2**	**0.4**	**0.6**	**0.8**	**0.2**	**0.4**	**0.6**	**0.8**	**0.2**	**0.4**	**0.6**	**0.8**	**0.2**	**0.4**	**0.6**	**0.8**	**0.2**	**0.4**	**0.6**	**0.8**
	# of Modes																				
PFPSO	**3**	1.79	2.08	0.51	0.10	na	1.50	0.89	0.20	na	na	1.33	0.36	na	na	na	0.51	na	na	na	0.48
	6	1.33	2.34	0.90	0.38	na	na	1.01	0.47	na	na	1.76	0.82	na	na	na	na	na	na	na	na
	9	1.25	2.33	0.94	0.49	na	na	0.94	0.44	na	na	na	0.82	na	na	na	na	na	na	na	na
CS-H-PM	**3**	0.17	0.04	0.05	0.03	na	0.13	0.11	0.06	na	na	0.20	0.04	na	na	na	0.15	na	na	na	0.13
	6	0.20	0.12	0.15	0.10	na	na	0.28	0.12	na	na	0.81	0.26	na	na	na	0.20	na	na	na	na
	9	0.26	0.24	0.23	0.10	na	na	0.26	0.12	na	na	na	0.27	na	na	na	na	na	na	na	na
CS-H-SM	**3**	0.17	0.04	0.06	0.01	na	0.12	0.07	0.04	na	na	0.16	0.05	na	na	na	0.15	na	na	na	0.15
	6	0.20	0.11	0.12	0.05	na	na	0.25	0.08	na	na	0.76	0.23	na	na	na	0.25	na	na	na	na
	9	0.26	0.23	0.17	0.10	na	na	0.25	0.12	na	na	na	0.30	na	na	na	na	na	na	na	na

Table 5.25 – Comparison of APD_{bin} values.

		APD_{bin} **(%)**																			
	# of Acts.	**50**				**100**				**200**				**500**				**990**			
	I_2	**0.2**	**0.4**	**0.6**	**0.8**	**0.2**	**0.4**	**0.6**	**0.8**	**0.2**	**0.4**	**0.6**	**0.8**	**0.2**	**0.4**	**0.6**	**0.8**	**0.2**	**0.4**	**0.6**	**0.8**
	# of Modes																				
PFPSO	**3**	1.79	1.73	0.32	0.01	na	1.24	0.59	0.07	na	na	0.66	0.07	na	na	na	0.09	na	na	na	0.03
	6	1.33	2.18	0.48	0.09	na	na	0.51	0.08	na	na	0.83	0.04	na	na	na	na	na	na	na	na
	9	1.25	2.12	0.54	0.14	na	na	0.38	0.11	na	na	na	0.05	na	na	na	na	na	na	na	na
CS-H-PM	**3**	0.18	0.02	0.01	0.00	na	0.10	0.06	0.00	na	na	0.10	0.00	na	na	na	0.02	na	na	na	0.01
	6	0.21	0.11	0.07	0.02	na	na	0.13	0.02	na	na	0.53	0.02	na	na	na	na	na	na	na	na
	9	0.27	0.22	0.12	0.05	na	na	0.14	0.06	na	na	na	0.03	na	na	na	na	na	na	na	na
CS-H-SM	**3**	0.18	0.02	0.00	0.00	na	0.10	0.03	0.00	na	na	0.09	0.00	na	na	na	0.02	na	na	na	0.01
	6	0.21	0.10	0.06	0.01	na	na	0.12	0.02	na	na	0.47	0.02	na	na	na	na	na	na	na	na
	9	0.27	0.20	0.11	0.05	na	na	0.11	0.04	na	na	na	0.03	na	na	na	na	na	na	na	na

Table 5.26 – Comparison of HR values.

		HR																			
	# of Acts.	**50**				**100**				**200**				**500**				**990**			
	I_2	**0.2**	**0.4**	**0.6**	**0.8**	**0.2**	**0.4**	**0.6**	**0.8**	**0.2**	**0.4**	**0.6**	**0.8**	**0.2**	**0.4**	**0.6**	**0.8**	**0.2**	**0.4**	**0.6**	**0.8**
	# of Modes																				
PFPSO	**3**	0.79	0.59	0.57	0.55	0.77	0.67	0.57	0.56	0.68	0.66	0.46	0.47	0.81	0.49	0.44	0.49	0.74	0.66	0.73	0.38
	6	0.86	0.67	0.64	0.57	0.84	0.75	0.59	0.58	0.71	0.79	0.55	0.36	0.88	0.72	0.91	0.00	0.80	0.93	0.74	0.00
	9	0.88	0.71	0.67	0.63	0.86	0.80	0.68	0.54	0.74	0.85	0.74	0.43	0.91	0.92	0.96	0.49	0.84	0.96	0.48	0.00
CS-H-PM	**3**	0.98	0.99	0.96	0.91	1.00	0.97	0.93	0.87	1.00	0.99	0.86	0.79	1.00	0.96	0.91	0.62	1.00	0.98	0.88	0.63
	6	0.98	0.98	0.90	0.86	1.00	0.99	0.81	0.72	1.00	0.99	0.71	0.49	1.00	0.79	0.72	0.83	1.00	0.74	0.66	0.96
	9	0.99	0.96	0.84	0.73	1.00	0.99	0.74	0.57	1.00	0.98	0.63	0.41	1.00	0.58	0.49	0.88	1.00	0.62	0.72	0.99
CS-H-SM	**3**	0.98	0.98	0.95	0.95	1.00	0.97	0.95	0.93	1.00	1.00	0.89	0.88	1.00	1.00	0.99	0.76	1.00	0.98	0.98	0.69
	6	0.98	0.98	0.93	0.88	1.00	1.00	0.88	0.82	1.00	0.99	0.90	0.71	1.00	0.97	0.87	0.88	1.00	0.88	0.72	0.99
	9	0.99	0.97	0.86	0.83	1.00	0.99	0.86	0.72	1.00	0.99	0.94	0.58	1.00	0.82	0.65	0.96	1.00	0.77	0.72	0.98

Table 5.27 – Comparison of CPU times.

CPU Time (s)																				
# of Acts.		**50**				**100**				**200**				**500**				**990**			
	I_2	**0.2**	**0.4**	**0.6**	**0.8**	**0.2**	**0.4**	**0.6**	**0.8**	**0.2**	**0.4**	**0.6**	**0.8**	**0.2**	**0.4**	**0.6**	**0.8**	**0.2**	**0.4**	**0.6**	**0.8**
# of Modes																					
PFPSO	**3**	0.06	0.04	0.02	0.01	0.45	0.35	0.16	0.07	2.18	2.99	0.97	0.22	59.68	11.65	8.07	2.87	222.09	131.69	55.08	18.41
	6	0.15	0.09	0.06	0.02	0.89	0.70	0.22	0.11	4.22	5.14	0.86	0.36	110.12	10.40	7.32	4.35	402.96	110.07	35.74	21.96
	9	0.21	0.14	0.10	0.06	1.19	0.94	0.36	0.24	5.32	5.61	1.14	0.94	134.76	11.64	8.33	7.59	482.16	111.85	45.65	39.67
CS-H-PM	**3**	0.06	0.04	0.02	0.01	0.45	0.35	0.16	0.07	2.18	2.99	0.97	0.22	59.68	11.65	8.07	2.87	222.09	131.69	55.08	18.41
	6	0.15	0.09	0.06	0.02	0.89	0.70	0.22	0.11	4.22	5.14	0.86	0.36	110.12	10.40	7.32	4.35	402.96	110.07	35.74	21.96
	9	0.21	0.14	0.10	0.06	1.19	0.94	0.36	0.24	5.32	5.61	1.14	0.94	134.76	11.64	8.33	7.59	482.16	111.85	45.65	39.67
CS-H-SM	**3**	0.06	0.03	0.03	0.32	0.46	0.34	0.16	1.30	2.17	2.93	1.20	10.09	58.79	11.85	8.54	107.61	218.20	131.32	48.02	838.27
	6	0.14	0.11	0.25	8.40	0.88	0.72	1.20	31.70	4.23	5.21	15.23	211.04	108.31	26.32	50.27	1847.83	392.90	141.49	146.22	8924.87
	9	0.20	0.25	0.97	18.61	1.18	1.06	5.81	77.00	5.32	7.24	58.34	427.82	133.36	76.16	182.71	3361.96	494.23	181.61	520.39	12973.30

The interpretation of the results tabulated in Table 5.22 to Table 5.27 can be summarized as follows. It is revealed that for the complete set of 50-activity problems, while being executed within the same timeframes, parallel variant of CS-Heuristic is able to capture 88.8% more non-dominated solutions than PFPSO. Per 120 problems of this size, parallel variant of CS-Heuristic with an average CPU time of 0.08 seconds, operates at least two seconds faster than the serial variant, on average; though, serial variant is observed to be able to achieve 4% more non-dominated solutions. On average, PFPSO, parallel, and serial variants are discovered to account for 1.7%, 71.5%, and 83.5% of the non-dominated solutions over the final unified Pareto front, respectively. Per 120 number of 50-activity problems, parallel and serial CS-Heuristics with comparable average deviations of 0.12% and 0.14%, are shown to be more accurate than PFPSO with an average APD of 1.2%. This remark is also true for the binary APD, since, PFPSO happens to have an average deviation of 1% from the optimal costs, for the mutually located duration amounts. This index for both CS-Heuristics is evaluated to be 0.1%. Solutions obtained by parallel and serial CS-Heuristics tend to be more accurate, well-distributed and widely spread with average HR values of 0.92 and 0.94; whereas this value for PFPSO is measured as 0.67.

Another interesting remark to note is that, for the 50-activity problems there is an increasing pattern in ONVG, APD, and APD_{bin} values of all the solution procedures with increasing mode numbers and decreasing I_2 parameters. This observation is also valid for the CPU times of the approaches, with the exception of serial variant of CS-Heuristic requiring more processing times for problems with larger I_2 values. For CS-Heuristics and PFPSO, HR values are determined to be larger for problems with smaller I_2s. Furthermore, HR values for PFPSO and CS-Heuristics are discovered to vary conversely for problems with I_2 values of 0.6 and 0.8. For these problems, HR of CS-Heuristics are observed to be inversely proportional to the number of modes, while, HR for PFPSO is directly related to the mode numbers.

Considering the whole set of 100-activity problems, parallel variant of CS-Heuristic is able to locate 96% more non-dominated solutions than PFPSO within the same execution time. Per 120 number of 100-activity problems, parallel variant of CS-Heuristic with an average CPU time of 0.47 seconds, operates at least nine seconds faster than the serial variant, on average; nevertheless, serial variant is observed to be able to obtain 6% more non-dominated solutions. On average, PFPSO, parallel, and serial variants are found out to account for 1%, 58.9%, and 80.5% of the non-dominated solutions located over the final unified Pareto front, respectively. Per 120 problems of this size, parallel and serial CS-Heuristics with comparable average deviations of 0.15% and 0.13%, are discovered to contribute to higher levels of exactness compared to PFPSO with an average APD of 0.77%. Regarding the average binary APD values, the above statement is also valid for the average deviations from the exact costs of same-duration solutions located by different methods. PFPSO happens to have an average APD_{bin} of 0.42%, whereas, this metric for parallel and serial CS-Heuristics are measured as 0.07% and 0.06%, respectively. Solutions obtained by CS-Heuristics – specifically the serial variant – are observed to be more accurate, well-distributed and widely spread, with higher average HR values. HR for PFPSO is evaluated to be 0.68, while, this value for parallel and serial CS-Heuristics are determined to be 0.88 and 0.92, respectively.

Moreover, an alternate conclusion can also be drawn for the 100-activity problems that there exists an increasing pattern in ONVG, APD, and APD_{bin} values of all the practiced methods with increasing mode numbers and decreasing I_2 rate. This observation is also valid for the computational times of the models, with the exception of serial variant of CS-Heuristic demanding significantly more CPU times for instances with larger I_2 values. For PFPSO, HR values are discovered to be inversely proportional to I_2 rate and directly proportional to the number of modes. For CS-Heuristics, however, HR values are determined to be inversely proportional to both I_2 parameter and the number of alternatives, with the

exception of problems with I_2 values of 0.2, all of which have HR values equal to 1.

It is identified that for the entire set of 200-activity problems, parallel variant of CS-Heuristic is capable of capturing 75.4% more non-dominated solutions than PFPSO by running within the same CPU time. For 120 problems with 200 activities, serial variant of CS-Heuristic is found out to require significantly longer runtimes. Compared to overall average of 2.5 seconds for parallel variant, serial variant is experimented to require a further one minute to locate the Pareto fronts; nonetheless, serial variant is revealed to be able to locate 7% more non-dominated solutions. On average, PFPSO, parallel, and serial variants are discovered to account for 2.8%, 47.6%, and 81.6% of the non-dominated solutions over the final unified Pareto front, respectively. Parallel and serial CS-Heuristics with literally the same average deviations of 0.31% and 0.3%, are conceded to be more accurate than PFPSO with an average APD of 1%. The same scheme is also confirmed for the binary APD, since, PFPSO happens to have an average APD_{bin} of 0.33%. This index for parallel and serial CS-Heuristics are assessed to be 0.13% and 0.12%, respectively. Solutions obtained by CS-Heuristics – especially the serial variant – are observed to be more accurate, well-distributed and widely spread, with greater average HR values of 0.82 and 0.9; however, HR for PFPSO is assessed to be 0.62.

Furthermore, another noteworthy conclusion can also be pointed out for the 200-activity problems that there exists an increasing pattern in ONVG, APD, and APD_{bin} values of all the exercised methods with increasing mode numbers and decreasing I_2 parameter. This remark is also true for the processing times of the methods, with the exception of serial variant of CS-Heuristic which operates within considerably larger CPU times for instances with larger I_2 values. The average HR values for all the approaches over the problems with I_2 values of 0.8 are observed to be inversely proportional to the number of modes. This is also true for parallel

variant of CS-Heuristic over the problems with I_2 values of 0.6. For PFPSO, HR values are discovered to be inversely proportional to I_2 rate and directly proportional to the number of modes over the problems with I_2s of 0.2 to 0.6. For CS-Heuristics, except for problems with I_2 of 0.6, HR values are determined to be either one or very close to one with no obvious trend in between.

For the complete set of 500-activity problems, parallel variant of CS-Heuristic is able to locate 39% more non-dominated solutions than PFPSO within the same time interval. Per 120 number of 500-activity problems, parallel variant of CS-Heuristic with an average CPU time of 31.39 seconds, runs within only 6.3% of the average computation time of serial variant; though, serial variant is observed to be able to capture 5% more non-dominated solutions. On average, PFPSO, parallel, and serial variants are discovered to provide 14.2%, 32.9%, and 68.3% of the non-dominated solutions positioned along the final unified Pareto front, respectively. Per 120 problems with 500 activities, parallel and serial CS-Heuristics with commensurate average deviations of 0.17% and 0.2%, are discovered to contribute to solutions of higher quality compared to PFPSO with an average APD of 0.51%. Unlike 50, 100, and 200 activity problems, average APD value for parallel variant is less than the serial CS-Heuristic for 500-activity instances. This trend is also confirmed for the binary APD, since, PFPSO happens to have an average deviation of 0.09% from the optimal costs, for the mutually located duration amounts. This index for both CS-Heuristics is calculated to be 0.02%. Solutions obtained by CS-Heuristics – particularly the serial variant – are observed to be more accurate, well-distributed and widely spread, with greater average HR values of 0.81 and 0.9; however, HR for PFPSO is measured to be 0.66.

Additionally, another remark can also be addressed for the 500-activity problems that there exists an increasing scheme for ONVG values of all the practiced methods with decreasing I_2 rate. For PFPSO, ONVG is observed to increase with the growth of mode numbers. For CS-Heuristics, however, the number of located non-

dominated solutions are discovered to decrease with the growth of the number of modes, with the exception of problems with I_2 s of 0.2. Since no 500-activity problems with I_2 values of 0.2 to 0.6 are solved to optimality, no clear pattern is captured for APDs and APD_{bin} s of these problems. However, it is observed that deviation amounts of CS-Heuristics are directly proportionate to the number of modes. Computational times of the models are addressed to have an increasing pattern with increasing mode numbers and decreasing I_2 parameter, with the exception of serial variant of CS-Heuristic for problems with I_2 rates of 0.6 and 0.8. Serial variant is experienced to require substantially greater runtimes for instances with larger I_2 values. While no clear scheme is detected for HR values of PFPSO, CS-Heuristics are observed to be inversely proportional to I_2 rate and the number of modes, except for the problems with I_2 values of 0.8 for which HRs increase with the growth of mode numbers. Besides, for both the CS-Heuristics, problems with I_2 values of 0.2 are shown to have HR values of one.

Lastly, it is acknowledged that for the whole set of 990-activity problems, parallel variant of CS-Heuristic is capable of obtaining 11% more non-dominated solutions than PFPSO by running within the same CPU time. For 120 problems including 990 activities, parallel variant of CS-Heuristic with an average processing time of 139.77 seconds, operates within only 6.7% of the average computation time of the serial variant. Despite its significantly larger computational burden, serial variant captures only 2% more non-dominated solutions than the parallel variant. On average, PFPSO, parallel, and serial variants are determined to deliver 14%, 44.3%, and 58% of the non-dominated solutions located along the final unified Pareto front, respectively. For 990-activity problems, parallel and serial CS-Heuristics with close average deviations of 0.13% and 0.15%, are noted to be more accurate than PFPSO with an average APD of 0.48%. Unlike 50, 100, and 200 activity problems, average APD value for parallel variant is smaller than the serial CS-Heuristic for 990-activity instances. This remark is also true for the binary APD, since, PFPSO

happens to have an average deviation of 0.03% from the exact costs, for the jointly located same-duration solutions. This index for both CS-Heuristics is measured to be 0.01%, on average. Solutions obtained by CS-Heuristics are validated to be more accurate, well-distributed and widely spread, with larger average HR values of 0.85 and 0.89. HR metric for PFPSO, on the other hand, is measured to be 0.6.

In addition, it can also be interpreted from the values of the performance metrics for the 500-activity problems that there exists an increasing scheme for ONVG values of all the practiced approaches with decreasing I_2 rate. For PFPSO, ONVG is discovered to increase with the growth of mode numbers. This is specifically true for problems with I_2 values of 0.2 and 0.6. For CS-Heuristics, however, the number of obtained non-dominated solutions are revealed to decrease with the growth of the number of modes, with the exception of problems with I_2s equal to 0.2. Inasmuch as no 990-activity problems with I_2 values of 0.2 to 0.6 are solved to optimality, no distinct pattern is captured for APDs and APD_{bin}s of these instances. Computational times of the models are determined to have an increasing pattern with increasing mode numbers. For problems with I_2 values of 0.2, processing time of all the models are directly proportionate to the number of modes. However, for the remaining I_2 values, there exists an inverse relation between the average CPU time and the number of modes, with the exception of serial variant of CS-Heuristic. Serial variant is experimented to contribute to remarkably larger computation times. While no clear pattern is identified for HR values, this index is noticed to be significantly lower for problems with I_2 values of 0.8 for PFPSO method. In addition, for problems with I_2 values of 0.2, CS-Heuristics are shown to be able to capture frontiers with HR values equal to one.

In the light of the above interpretations, an overall picture of PFPSO's and CS-Heuristic's performances can be obtained as follows. For the entire collection of the RanGen2 instances, both variants of CS-Heuristic are shown to be able to achieve

higher number of Pareto solutions, serial variant even more so than the others. It is also discovered that the number of non-dominated solutions located along the final unified frontier are significantly larger for CS-Heuristics. In fact, PFPSO only accounted for less than 7% of these solutions, while serial variant positioned 23% more than the parallel CS-Heuristic. Despite high levels of accuracy for all the experimented methods, CS-Heuristics are confirmed to be more successful in converging to the true Pareto fronts with fractional deviations. Average deviations for parallel and serial CS-Heuristic are shown to be literally the same; whereas, PFPSO's average deviation value is calculated to be five to six times the amount for CS-Heuristics. Well-distributed and widely spread fronts are only obtained by means of CS-Heuristic since the area of the solution space covered by PFPSO, poorly represents the Hypervolume of the true/best Pareto front.

Although the values for ONVG, ND_{pct}, and HR performance metrics are sporadically higher for the serial variant of CS-Heuristic, it is experienced to contribute to remarkably higher computational costs. In fact, serial variant is conceded to operate within an average of 15 times the CPU time of parallel CS-Heuristic. The reason behind this major increase in CPU time is due to the enormously large number of serially merged activities for the problems with pseudo-serial networks. It is observed that for higher I_2 values, viz., 0.6 and 0.8, the number of serial activities satisfying the conditions for the serial merge can reach up to almost 50% of the original network size. For instance, some 990-activity problems are observed to be reduced by merging 487 number of serial activities. As is clear, this technique is not suitable for networks with serial graphs and that the marginal improvements in the obtained solutions might not justify the additional processes and the extra efforts involved in merging the activities serially.

Inclusion of parallel merge, on the other hand, is experimented to contribute to lower computational times, although by meager amounts, with the exception of problems with 500 and 990 activities and larger I_2 parameters. Parallel merge is

discovered to slightly increase the computation time for 500-activity problems, and even more so for 990-activity instances, with I_2 values of 0.8. The reason behind this slight growth in computation time is due to surging number of parallel reducible activities. As mentioned in Section 5.1.1.1, the first step for parallel merging is defined as to search for activities including only two successors. For the problems generated by means of RanGen2, the pseudo-parallel networks with smaller I_2 values comprise a multitude of successors (generally more than two); however, as the value of I_2 increases, the number of successors for the activities of pseudo-serial networks reduce to either one or two. That is the reason why a large percentage of the activities on the pseudo-serial networks satisfy the conditions for parallel merging. Meanwhile, it is also observed that both the serial and parallel merging techniques practically increase ONVG, ND_{pct}, and HR values for problems with 500 and 990 activities. Furthermore, for the new sets of RanGen2 instances, implementation of Partial-CPM technique (Section 5.1.2) is experienced to reduce the overall average computation time of CS-Heuristic by 5.2%.

With respect to the number of the obtained non-dominated solutions and the diversity of the captured Pareto fronts, the nature of PFPSO and CS-Heuristic approaches, in contrast to MILP, appear to be more suitable for pseudo-parallel networks with smaller I_2 values that include greater number of time-cost alternatives. However, for this setting of problems the deviation and the computation time amounts are observed to increase, although by meager amounts. Comparative studies reveal that the performance of CS-Heuristic is unmatched by any of the previous approaches including the PFPSO algorithm. Owing to the unprecedented accuracy and diversity of the obtained Pareto fronts, as well as its exceptional efficiency, Cost-Slope Heuristic is expected to contribute to optimal planning of realistic large-scale construction projects. To the best of author's knowledge, the proposed CS-Heuristic optimization model is the first method that outperforms highly capable meta-heuristics and is able to tackle large-scale problems consisting of hundreds of activities that are based upon the complex

RanGen2 networks, within reasonably short extents of time and practically viable deviations.

5.2.5.5. Case-Problems

Robustness of the proposed PFPSO and CS-Heuristics for Pareto front DTCTP are also validated using two real construction projects. The proposed solution procedures are tested on the same desktop computer. The proposed approaches are first experimented using a warehouse construction project which is obtained from Chen and Weng (2009). This project consists of 37 activities with up to two time-cost alternatives. The indirect cost rate is \$600/day, the delay penalty is set as \$500/day, and the completion deadline is assumed as 240 days. The exact results of MILP (Section 5.2.3) and the approximations of CS-Heuristics (Section 5.1.4) and PFPSO (Section 4.3) are presented in Table 5.28.

Table 5.28 – Comparison of the results for the first case problem.

Algorithm	**ONVG**	ND_{pct} **(%)**	**APD (%)**	APD_{bin} **(%)**	**HR**	**CPU Time (s)**
MILP (Section 5.2.3)	1					0.06
PFPSO (Section 4.3)	1	100	0	0	1	0.02
CS-Heuristic (Section 5.1.4) with PM*	1	100	0	0	1	0.02
CS-Heuristic (Section 5.1.4) with SM**	1	100	0	0	1	0.02

*Parallel Merge

**Serial Merge

As shown in Table 5.28, all the proposed approaches are capable of capturing the true Pareto front of this real project which includes a single solution with a duration of 174 days and an overall cost of \$253,400. The solution times of parallel and serial CS-Heuristics and PFPSO are discovered to be the same.

The second real construction project used for experimentation of the proposed approaches include a process plant project which is acquired from Abbasi-Iranagh (2015). This project consists of 519 activities with up to four time-cost alternatives.

The indirect cost rate is $10,000/day, the delay penalty is assumed as $20,000/day, and the completion deadline is defined to be 540 days. Performance of results for MILP (Section 5.2.3), CS-Heuristics (Section 5.1.4), and PFPSO (Section 4.3) are summarized in Table 5.29.

Table 5.29 – Comparison of the results for the second case problem.

Algorithm	**ONVG**	ND_{pct} **(%)**	**APD (%)**	APD_{bin} **(%)**	**HR**	**CPU Time (s)**
MILP (Section 5.2.3)	1					1.10
PFPSO (Section 4.3)	2	0	1.09	1.09	0.34	0.07
CS-Heuristic (Section 5.1.4) with PM*	1	100	0.43	0.43	0.72	0.07
CS-Heuristic (Section 5.1.4) with SM**	1	0	0.49	0.49	0.69	0.13

*Parallel Merge

**Serial Merge

As shown in Table 5.29, MILP is able to capture the true Pareto front for the second real project in 1.1 seconds. The optimal Pareto front for this project comprise a unique non-dominated solution with a duration of 626 days and an overall cost of $12,620,417. Being executed within the same computational time, PFPSO is discovered to be able to locate two non-dominated solutions compared to a single non-dominated solution of all the other approaches. PFPSO, despite contributing to a greater ONVG value, is observed to provide solutions with slightly higher deviations. The area of the solution space covered by PFPSO is also observed to poorly represent the Hypervolume of the true Pareto front. Parallel CS-Heuristic is shown to be able to provide the best approximate Pareto front for this project with the greatest ND_{pct}, smaller unary and binary deviation, and larger HR by running within a shorter timeframe.

Based on the extremely successful performance of the proposed algorithms for the real projects, implications can be drawn out as the realistic projects tend to be relatively simpler than the instances generated in Section 5.2.1. The deviation and the runtime amounts are conceded to be small even for a large-scale 519-activity

construction project. According to the results obtained for the two real construction projects, practicability and real-life applicability of the proposed approaches are firmly validated. The proposed optimization models enjoy the fastness and are able to capture optimal/near optimal sets of non-dominated solutions for the multi-objective DTCTP. The author proposes to employ the proposed CS-Heuristic and PFPSO methods in place of the MILP when the optimality is not regarded as the most crucial concern.

CHAPTER 6

INTEGRATION OF THE PROPOSED METHODS INTO MICROSOFT PROJECT

Despite the fact that any scientific decision support tool would have a pivotal role in the decision-making process, none of the commercial scheduling software packages (e.g., Microsoft Project, Primavera) include tools or modules for time-cost trade-off analyses of the scheduling problems. To the respect of this, this chapter is dedicated to the development of a tool which bridges the gap between the scheduling software and the DTCTP optimization techniques. Therefore, the proposed discrete time-cost trade-off problem optimization algorithms are integrated into the Microsoft Project 2013 which is a widely used commercial planning software in the construction industry. Integration is facilitated by means of an add-in which is capable of solving two variants of DTCTP, namely, cost minimization/deadline and Pareto front problems. The integrated modules include the proposed CS-Heuristics (Section 5.1.3 and Section 5.1.4), DPSO (Section 3.3), and PFPSO (Section 4.3) approaches which are integrated by means of an add-in implemented in C# programming language using Microsoft Visual Studio 2013. By means of the built add-in, users will readily be able to visualize the optimized schedules for their projects.

After installation of the created add-in, a ribbon named “TCTP” will appear among the existing tabs of the Microsoft Project. Selecting this new tab will direct users to a menu which includes two groups of “Heuristic” and “Meta-Heuristic”. Either of the mentioned groups comprise two buttons of “Optimize” and “Pareto front”.

As the names imply, CS-Heuristic methods are classified under “Heuristic”, while DPSO and PFPSO are placed under “Meta-Heuristic” group. The “Optimize” button under “Heuristic” group invokes CS-Heuristic for deadline DTCTP (Section 5.1.3) while “Pareto front” button recalls the CS-Heuristic for Pareto front DTCTP (Section 5.1.4). Likewise, the “Optimize” button under “Meta-Heuristic” group invokes DPSO (Section 3.3) while “Pareto front” button activates the PFPSO (Section 4.3) module. As displayed in Figure 6.1, a “Quick-Solve” option is also included under the “Meta-Heuristic” group on the main panel. When checked, this option avails PFPSO module to run up to five times faster by adjusting the parameters of this module. The user interface (UI) of the developed add-in is illustrated in Figure 6.1.

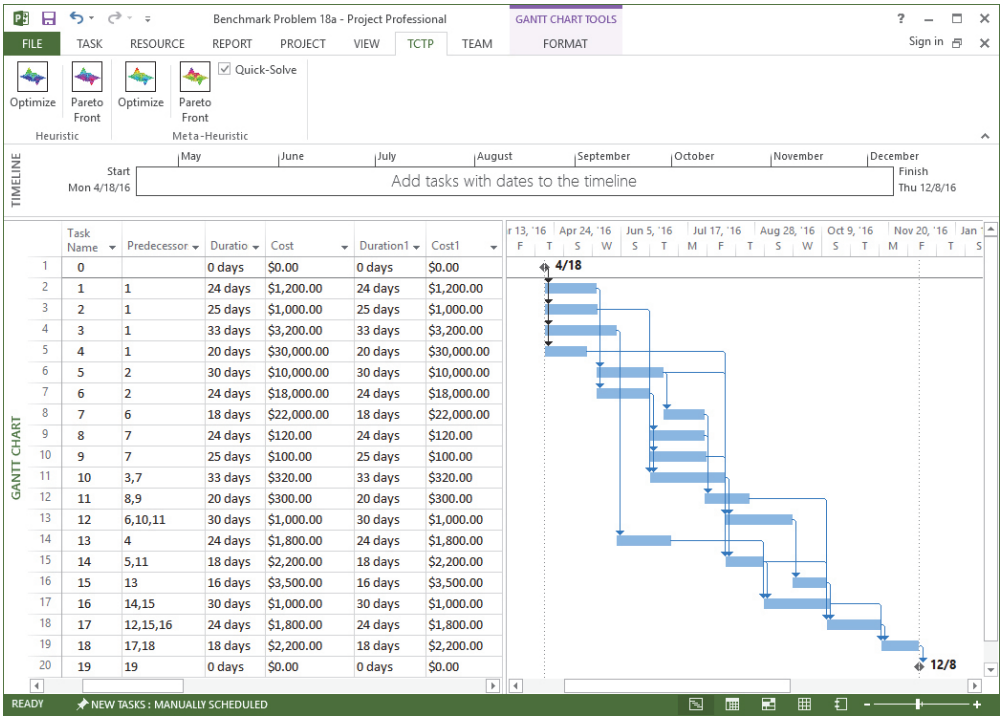
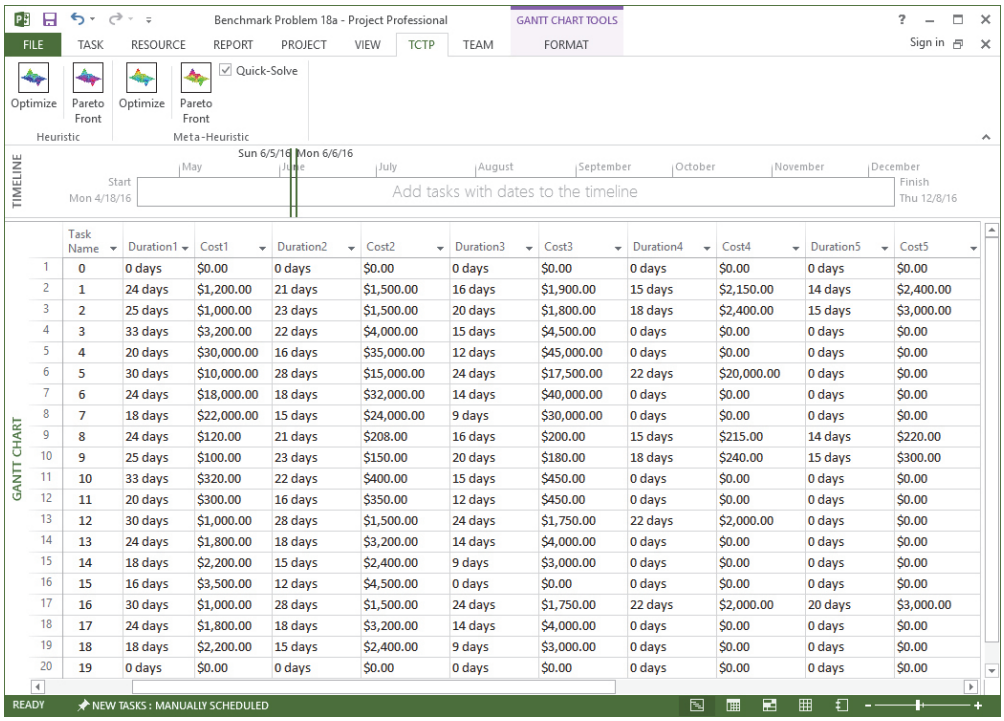


Figure 6.1 – User Interface of the Microsoft Project Add-in.

18a problem which is introduced in Section 3.4.2 is implemented to exemplify the application of the designed TCTP add-in. Precedence relationships are defined for

the project activities through the standard procedure. As shown in Figure 6.2, time-cost alternatives of the activities are entered into the corresponding fields as follows. For each activity, “Duration 1” and “Cost 1” are filled in with the time and cost of the normal mode, respectively. That is, modes with the longest durations and the least direct costs are entered first. The remaining time-cost alternatives are entered into the succeeding cells (i.e., “Duration 2”/”Cost 2” to “Duration 10”/”Cost 10”) in ascending order with regard to their direct cost figures. By default, Microsoft Project holds “0 days” and “$0.00” for any unfilled entries of time and cost, respectively. Hence, for the activities with fewer options than the maximum supported number of modes, i.e., ten, the extra fields are left unmodified.



	Task Name	Duration1	Cost1	Duration2	Cost2	Duration3	Cost3	Duration4	Cost4	Duration5	Cost5
1	0	0 days	$0.00	0 days	$0.00	0 days	$0.00	0 days	$0.00	0 days	$0.00
2	1	24 days	$1,200.00	21 days	$1,500.00	16 days	$1,900.00	15 days	$2,150.00	14 days	$2,400.00
3	2	25 days	$1,000.00	23 days	$1,500.00	20 days	$1,800.00	18 days	$2,400.00	15 days	$3,000.00
4	3	33 days	$3,200.00	22 days	$4,000.00	15 days	$4,500.00	0 days	$0.00	0 days	$0.00
5	4	20 days	$30,000.00	16 days	$35,000.00	12 days	$45,000.00	0 days	$0.00	0 days	$0.00
6	5	30 days	$10,000.00	28 days	$15,000.00	24 days	$17,500.00	22 days	$20,000.00	0 days	$0.00
7	6	24 days	$18,000.00	18 days	$32,000.00	14 days	$40,000.00	0 days	$0.00	0 days	$0.00
8	7	18 days	$22,000.00	15 days	$24,000.00	9 days	$30,000.00	0 days	$0.00	0 days	$0.00
9	8	24 days	$120.00	21 days	$208.00	16 days	$200.00	15 days	$215.00	14 days	$220.00
10	9	25 days	$100.00	23 days	$150.00	20 days	$180.00	18 days	$240.00	15 days	$300.00
11	10	33 days	$320.00	22 days	$400.00	15 days	$450.00	0 days	$0.00	0 days	$0.00
12	11	20 days	$300.00	16 days	$350.00	12 days	$450.00	0 days	$0.00	0 days	$0.00
13	12	30 days	$1,000.00	28 days	$1,500.00	24 days	$1,750.00	22 days	$2,000.00	0 days	$0.00
14	13	24 days	$1,800.00	18 days	$3,200.00	14 days	$4,000.00	0 days	$0.00	0 days	$0.00
15	14	18 days	$2,200.00	15 days	$2,400.00	9 days	$3,000.00	0 days	$0.00	0 days	$0.00
16	15	16 days	$3,500.00	12 days	$4,500.00	0 days	$0.00	0 days	$0.00	0 days	$0.00
17	16	30 days	$1,000.00	28 days	$1,500.00	24 days	$1,750.00	22 days	$2,000.00	20 days	$3,000.00
18	17	24 days	$1,800.00	18 days	$3,200.00	14 days	$4,000.00	0 days	$0.00	0 days	$0.00
19	18	18 days	$2,200.00	15 days	$2,400.00	9 days	$3,000.00	0 days	$0.00	0 days	$0.00
20	19	0 days	$0.00	0 days	$0.00	0 days	$0.00	0 days	$0.00	0 days	$0.00

Figure 6.2 – Defining time-cost alternatives for Microsoft Project Add-in.

As shown in Figure 6.3, by clicking on any of the buttons, “Project Details” window pops-up which prompts users to enter the rate of the indirect cost ($/day), project deadline (days), and the amount of delay penalty ($/day).

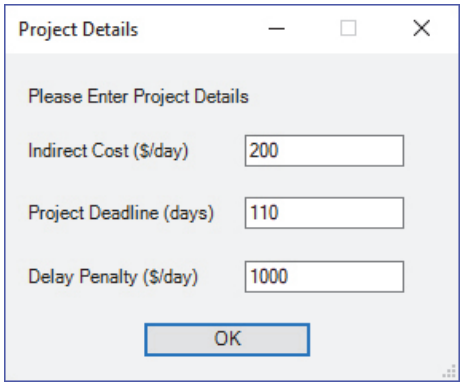


Figure 6.3 – Project Details window of Microsoft Project Add-in.

In case the cost minimization/deadline problem is practiced by means of either CS-Heuristic or DPSO modules, TCTP add-in, after acquiring the project information proceeds to a new window (Figure 6.4) which displays the unique optimal solution for the single-objective time-cost trade-off problem.

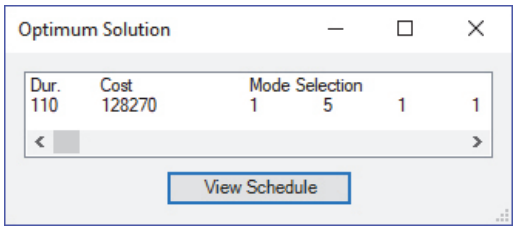


Figure 6.4 – Optimal solution window of Microsoft Project Add-in.

By pressing the "View Schedule" button on "Optimum Solution" window, users are directed to the optimal solution which demonstrates the optimal selection of the time-cost options that yields the optimum schedule. Information on the generated schedule, including early dates, late dates, and floats, can be obtained from Microsoft Project which also provides the visual view of the activities in the form of a Gantt chart (Figure 6.5).

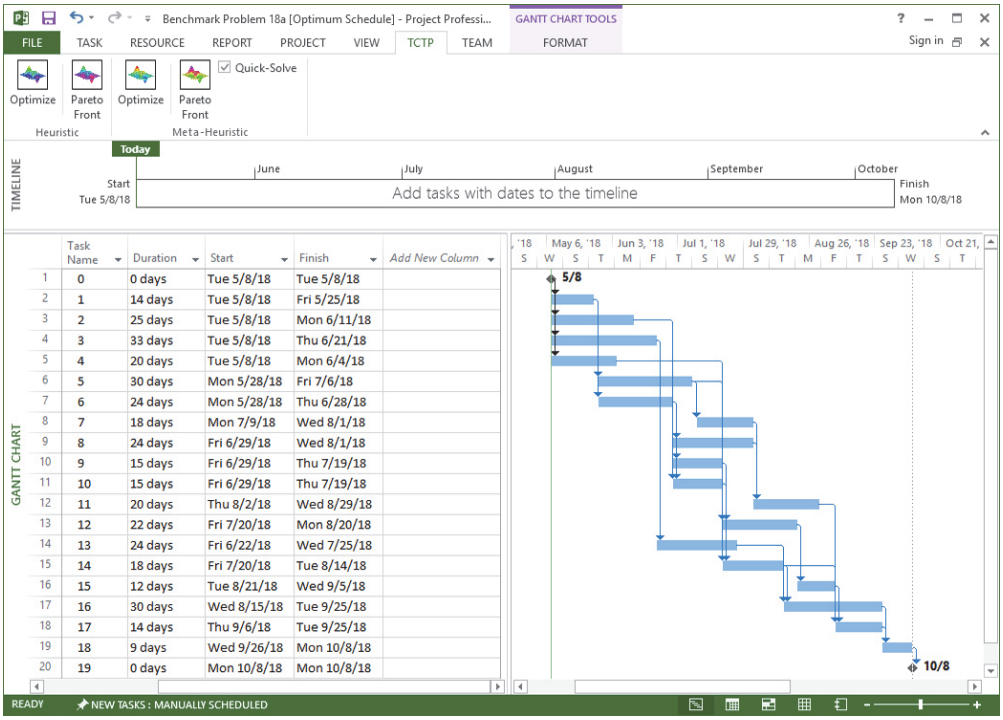


Figure 6.5 – Optimal schedule generated by Microsoft Project Add-in.

In case the Pareto front problem is practiced for the active project by means of either CS-Heuristic or PFPSO modules, TCTP add-in, after acquiring the project information proceeds to a new window which is illustrated in Figure 6.6. The "Pareto Front Solutions" window provides the sequence of the non-dominated solutions which are listed in ascending order with regard to duration amounts.

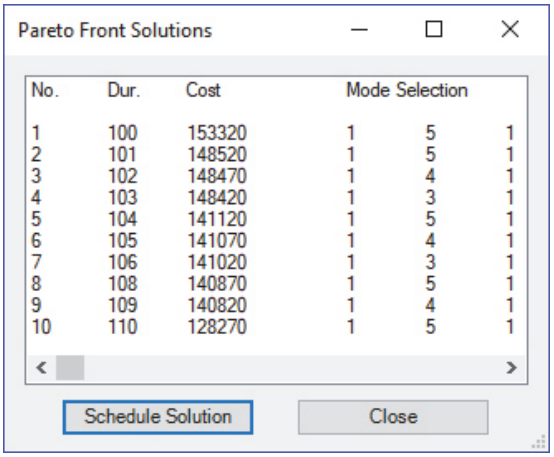


Figure 6.6 – Pareto front solutions window of Microsoft Project Add-in.

The "Pareto Front Solutions" window prompts users to select the desired non-dominated solution(s) from the schedules listed (Figure 6.7).

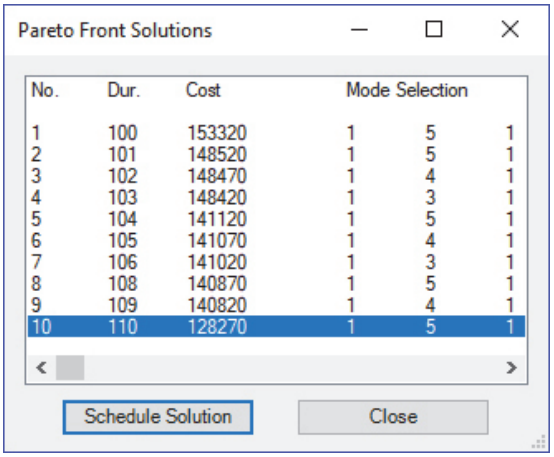


Figure 6.7 – Selection of a non-dominated solution achieved by Microsoft Project Add-in.

By pressing the "View Schedule" button for each of the selected solutions listed on "Pareto Front Solutions" window, users are directed to each non-dominated solution individually which reveal the arrangements of time-cost alternatives for the desired schedules. Details on the generated schedule, including early dates, late

dates, and floats, can be attained from Microsoft Project which also presents a visual view of the project activities in the form of a Gantt chart (Figure 6.8).

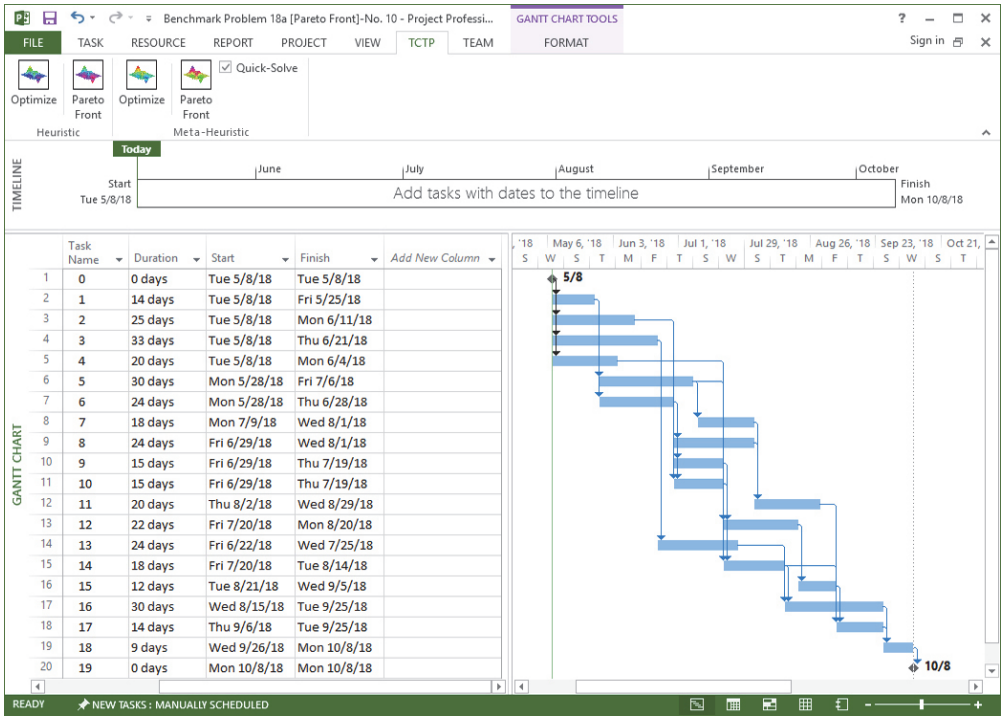


Figure 6.8 – Schedule for the selected Pareto solution generated by Microsoft Project Add-in.

TCTP add-in is acknowledged to enhance the practicability of the proposed DTCTP optimization algorithms. Benefiting from the presented add-in, users of Microsoft Project will readily be able to visualize the optimized schedules for the practiced projects; hence, the proposed methods are supposed to be more readily accepted and used by the parties to construction projects. By means of the developed add-in, the new optimization models are envisioned to be applicable in real projects and to suit the actual practices of construction managers. It is expected that these approaches might prove to be an efficient and effective base for exerting this highly challenging problem.

CHAPTER 7

CONCLUSIONS

The significance of the counteracting aspects of time and cost for construction projects is highly emphasized since project success is chiefly related to these factors. A key process for efficacious realization of the anticipated outcomes of time and cost is acknowledged to be the preparation of exhaustive plans and impeccable schedules. It is declared that construction projects are prone to major financial losses short of optimal schedules. Of the schedule optimization methodologies, exertion of time-cost trade-off problem is reckoned to play a crucial role in securing the pre-specified objectives of time and cost. It is a problem solving and decision-making science which provides the management with a quantitative basis for decisions on selection of the optimal time-cost alternatives. Despite its widely accepted practical significance, no major real-life applicable approach is discovered within the construction management literature. The author criticizes a large body of the existing research due to the insufficient details on test configurations as well as the inadequate size of the practiced problems. The literature on large-scale time-cost trade-off problem is discovered to be virtually void, with only a few studies using large-scale problems that are generated by cloning simple instances. In spite of practicability of discrete multi-objective variant of the time-cost trade-off problem, no real-life applicable contribution is observed within the earlier research. A few methods that are implemented for real-life-size large-scale problems, are conceded to lead to unrealistically significant computational efforts. Though, for any solution method to be practically viable, accuracy needs to be accompanied with the efficiency. Notwithstanding the fact

that achieving an adequate schedule boils down to utilization of scientific decision support tools, no commercial scheduling software provides tools for time-cost trade-off analyses. Accordingly, in order to bridge the gap between the theoretical and practical relevance in time-cost trade-off problem, new exact, heuristic, and meta-heuristic approaches with sound convergence capabilities are proposed, some of which are integrated to a popular scheduling software package within the context of this thesis.

Apart from the fact that the existing approaches have seldom been applied for solution of large-scale DTCTPs, it is interpreted that the dearth of real-life-scale problems could possibly be another major reason for the lack of studies on realistic problems. Despite the fact that some studies have included problems including up to 720, 2000, and 6300 activities, all of the employed large-scale problems are generated using small-scale base networks and are generated by copying the core problem in serial several times; hence, these problems are anticipated to have limitations in reflecting the complexity of the real-life construction projects. In order to have a better understanding of the behavior and capabilities of the proposed optimization models, in addition to the existing benchmark and case problem, new sets of multi-mode large-scale DTCT problems including up to 990 activities have been generated by means of RanGen2 random network generator. The systematically generated large-scale instances comprise complex networks and are treated with realistic sets of time-cost alternatives. In order to compare methods rigorously and to quantitatively measure performance of different approaches over the benchmark and the RanGen2 instances, performance metrics are employed. The incorporated performance comparison indices are designed to measure cardinality, accuracy, diversity, and efficiency of the optimization models. Accuracy-based performance indices necessitate acquisition of exact optimal solutions; to this end, an exact optimization method is developed.

The proposed exact method is based on Mixed-Integer Linear Programming which engages Gurobi solver. Different variants of this method are introduced for two paradigms of cost minimization/deadline and Pareto front discrete time-cost trade-off problems. Since, exact procedures are the only methods guaranteeing optimality of the solutions, the proposed exact methods are mainly used for validation of the performance of the developed heuristic and meta-heuristic approaches. The proposed exact methods are equipped with a new merging technique which exponentially decreases the scale of the practiced problems. The multi-objective variant of the proposed exact method is also equipped with an efficient upper-bound calculator designed to reduce the size of the solution space. In the light of the implemented techniques, computation time of the exact methods are significantly reduced. The presented single-objective exact method is shown to be able to successfully converge to optimal solutions for 100% of 50-activity, 100% of 100-activity, 71.66% of 200-activity, 49.16% of 500-activity, and 25% of 990-activity deadline DTCTPs within the enforced runtime limit of one hour. Similarly, the multi-objective exact method is experimented to effectively capture true Pareto fronts for 95.83% of 50-activity, 51.66% of 100-activity, 35.83% of 200-activity, 15% of 500-activity, and 8.3% of 990-activity Pareto front DTCTPs within the imposed CPU time limit of one hour. It is interpreted that the natures of these methods tend to be more suitable for pseudo-serial networks that include smaller number of time-cost alternatives. Implementation of the upper-bound and the merging techniques are confirmed to increases the number of solved problems by more than 19% and to reduce the overall average CPU time by more than 18%. To the best of author's knowledge, this is the first contribution where global optimal costs and true Pareto fronts are captured for real-life-scale instances that are based upon the complex RanGen2 networks.

It is alleged that the existing literature is not rich with particle swarm optimization exemplars with the capacity to tackle realistic large-scale DTCT problems. To the respect of this, different PSO algorithms are proposed for two extensions of cost

minimization/deadline and Pareto front DTCT problems. Both the introduced PSOs are equipped with unique semi-deterministic initialization techniques and use new principles for presentation and position-updating of the particles. The discrete PSO which is designed for the single-objective DTCTP, is complemented with the modified-SAM heuristic; whereas, the multi-objective Pareto front PSO is enhanced using the simplified Heuristic. The trajectories and velocities of the presented PSO-based approaches are defined as probabilities.

The solid convergence capabilities of DPSO is first validated for solution of small, medium, and large-scale benchmark problems as well as new sets of instances generated by means of ProGen/Max random instance generator. Later, its performance is measured against the proposed Cost-Slope Heuristic. The comparison of DPSO with the state-of-the-art methods proved that DPSO is among the best, if not the best, meta-heuristic approach for single-objective DTCTP with respect to both solution quality and computation time. DPSO is shown to be able to produce good feasible solutions in acceptable timeframes even for the complex ProGen/Max instances. An average deviation of 0.21% from the optima of 500-activity problems is considered to be practically reasonable which establishes DPSO as an effective and robust alternative for real-world applications. To the best of author's knowledge, the proposed DPSO is one of the first methods capable of obtaining high quality solutions for the large-scale single-objective DTCTPs within seconds.

The sound convergence capabilities of PFPSO is first illustrated for solutions of benchmark problems attained from the literature. Later, its performance is compared to the proposed Cost-Slope Heuristic using the new sets of RanGen2 instances as well as case problems. The computational tests involving benchmark problems revealed that the proposed PFPSO can provide significantly larger number of non-dominated solutions for small, medium, and large-scale problems, and remarkably outperformed the well-developed methods. The results revealed

that the computation time requirement of PFPSO is considerably less than that of the existing methods. It is demonstrated that PFPSO is capable of locating high quality non-dominated solutions which are either optimal or very close to the optimal costs. To the best of author's knowledge, this is one of the first contributions where a meta-heuristic algorithm is able to adequately solve real-life-scale multi-objective DTCTPs within seconds, a performance which is unmatched by the previous meta-heuristic methods.

It is broadly acknowledged that evolutionary algorithms are very sensitive to configuration of their parameters. In real-life situations, the experimental process for configuration of parameters for best values may become a tedious and arduous task. Yet, the parameters need to be retuned for each new problem at hand which might reduce the practicability of the meta-heuristic approaches. The proposed DPSO and PFPSO methods are not exceptions to this. On the other hand, experimentation of the PSO-based approaches revealed that their exceptional performances were largely resulting from their heuristic modules. It was also observed that none of the previous heuristics have the capacity to tackle large-scale Pareto front DTCT problems. Consequently, regarded as the chief contribution of this thesis, different variants of a new Cost-Slope Heuristic are designed and developed within the context of this thesis. The proposed CS-Heuristics include parallel and serial versions for both cost minimization/deadline and Pareto front classes of DTCTPs. CS-Heuristic engages unique scientific and programmable rules comprising an innovative Partial-CPM technique which is designed to accelerate the solutions process. Furthermore, similar to the proposed exact methods, parallel and serial merging techniques are implemented to reduce the scale and computation cost of the practiced problems.

Comparative studies on the single-objective CS-Heuristic involving a set of small, medium, and large-scale benchmark problems not only confirmed its soundness, but also revealed its superiority over earlier state-of-the-art approaches. While it is

able to locate high quality solutions for all the practiced cost minimization/deadline problems, computation time requirement of this method is also demonstrated to be remarkably less than the earlier approaches. In fact, by running on the same desktop computer, the processing time of this method is experimented to be less than that of DPSO. It is proved that the proposed CS-Heuristic by outperforming the highly capable DPSO method, can converge to global optimal solutions with only fractional deviations. To the best of author's knowledge, the proposed CS-Heuristic optimization model is the first method that outdoes state-of-the-art meta-heuristic approaches and is capable of unraveling large-scale problems comprising thousands of activities within practically reasonable timeframes with only fractional deviations.

Comparative studies on the multi-objective CS-Heuristic involved a set of existing small, medium, and large-scale benchmark instances, case problems acquired from the literature, and new sets of RanGen2 instances. A collection of unary and binary performance metrics was measured in the course of performance evaluations including cardinality, accuracy, diversity, and efficiency indices. Results revealed an unmatched performance by CS-Heuristic in comparison with the previous approaches including PFPSO algorithm. It is concluded that not only the computation time requirement of the innovative multi-objective CS-Heuristic is substantially less than the earlier approaches, but it is also able to produce a large number of high quality non-dominated solutions for all the practiced Pareto front problems. Compared to results of PFPSO over the entire collection of the RanGen2 instances, both parallel and serial variants of CS-Heuristic are shown to be able to achieve higher number of Pareto solutions, serial variant even more so than the others. It is also discovered that the number of non-dominated solutions located along the final unified frontier are significantly larger for CS-Heuristics. In fact, PFPSO only accounted for less than 7% of these solutions, while serial variant captured 23% more than the parallel CS-Heuristic. Despite high levels of accuracy for all the experimented methods, CS-Heuristics are confirmed to be more

successful in converging to the true Pareto fronts with fractional deviations. Average deviations for parallel and serial CS-Heuristic are shown to be literally the same; whereas, PFPSO's average deviation value is calculated to be five to six times the amount for CS-Heuristics. Well-distributed and widely spread fronts are only obtained by means of CS-Heuristic since the area of the solution space covered by PFPSO, poorly represents the Hypervolume of the true/best Pareto front. It is also interpreted that the nature of PFPSO and CS-Heuristic approaches, in contrast to the proposed exact method, tend to be more suitable for pseudo-parallel networks that include greater number of time-cost alternatives. To the best of author's knowledge, the proposed CS-Heuristic optimization model is the first method that outperforms the highly capable meta-heuristic approaches and is able to tackle large-scale problems consisting of hundreds of activities within reasonably short timespans and practically viable deviations. Owing to its unprecedented efficacy and exceptional accuracy, Cost-Slope Heuristic is expected to contribute to optimal planning of realistic construction projects.

Practicability and real-life applicability of the proposed PFPSO and CS-Heuristics were firmly validated by means of real construction projects acquired from the literature. Due to the extremely successful performance of the proposed optimization approaches over the real projects, it is interpreted that the realistic projects tend to be relatively simpler than the instances generated within the course of this thesis. The deviation and the runtime amounts are conceded to be small even for the large-scale 519-activity construction project. In fact, all the methods are observed to be able to locate high quality solutions which are either optimal or very close to the optimal frontier within less than a second.

Integration of the proposed optimization algorithms into Microsoft Project is also presented in this thesis. An add-in which is capable of solving cost minimization/deadline and Pareto problems is developed which includes both the PSO-based methods and the CS-Heuristics. It is acknowledged to enhance the

practicability of the proposed DTCTP optimization algorithms. Benefiting from the presented add-in, users of Microsoft Project will readily be able to visualize the optimized schedules for the practiced projects; hence, the proposed methods are supposed to be more readily accepted and used by the parties to construction projects. By means of the developed add-in, the new optimization models are envisioned to be applicable in real projects and to suit the actual practices of construction managers. It is expected that these approaches might prove to be a robust base for exerting this highly challenging problem.

To wrap up, all the different proposed DTCTP optimization models are shown to be innovational and efficacious which are highly relevant for real-life applications. Particularly, the CS-Heuristic is regarded as a pioneering method which presents a new uncrashing concept along with salient techniques for network reduction and faster analysis of the networks. Experimentations attest to the efficacy and efficiency of the proposed methods for successful solution of real-life-scale problems. While the multi-objective variants of all the presented models enable articulation of decision makers' preferences, the author proposes to employ the suggested heuristic or meta-heuristic approaches in place of the described exact method when the optimality is not regarded as the most crucial concern.

The proposed models still have some limitations which are to be addressed in future studies. Firstly, all the developed methods are designed to solve problems with standard networks. They can only tackle networks with logical relationships of type finish-to-start, considering no lags in between. However, the proposed models should consider generalized precedence relationships by covering all the other types of constraints including finish-to-finish, start-to-finish, and start-to-start logical relationships. Besides, they should also allow for inclusion of positive and negative lag times between the activities. Secondly, the proposed Microsoft Project add-in accepts only up to ten time-cost alternatives. Though, it is possible for a project

activity to comprise more than ten options; hence, the proposed add-in should support a larger number of entries for time-cost options.

Despite exceptional performance of the proposed models, there still remains room for improvements. The proposed models run on a single core of the CPU. Though, dividing the parallel calculations and processes of the algorithms into several cores would further enhance the convergence speed of the optimization approaches. Much research also remains to be done toward extending the developed models to incorporate other aspects of construction projects such as uncertainties, resource limitations, safety, productivity, and quality. Last but not least, development of a hybrid meta-heuristic algorithms by capitalizing on the solid convergence capabilities of the proposed CS-Heuristic appears to be a promising research area.

REFERENCES

Abbasi-Iranagh, M. (2015). *Development of High Performance Heuristic and Meta-Heuristic Methods for Resource Optimization of Large Scale Construction Projects*. Doctoral dissertation, Middle East Technical University, Ankara, Turkey.

Abdel-Raheem, M., & Khalafallah, A. (2011). Using Electimize to Solve the Time-Cost-Tradeoff Problem in Construction Engineering. *Proc., ASCE Int. Workshop on Computing in Civil Engineering, ASCE*, pp. 250–257, Reston, VA.

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-ASCE*, *135*(7), 668-674.

Agdas, D., Warne, D. J., Osio-Norgaard, J., & Masters, F. J. (2018). Utility of Genetic Algorithms for Solving Large-Scale Construction Time-Cost Trade-Off Problems. *Journal of Computing in Civil Engineering*, *32*(1), 04017072.

Agrawal, M. K., Elmaghraby, S. E., & Herroelen, W. S. (1996). DAGEN: A Generator of Testsets For Project Activity Nets. *European Journal of Operational Research*, *90*(2), 376–382.

Akkan, C., Drexl, A., & Kimms, A. (2005). Network Decomposition-Based Benchmark Results for the Discrete Time-Cost Tradeoff Problem. *European Journal of Operational Research*, *165*, 339-358.

Aminbakhsh, S. (2013). *Hybrid Particle Swarm Optimization Algorithm for Obtaining Pareto Front of Discrete Time-Cost Trade-Off Problem*. Master's thesis, Middle East Technical University, Ankara, Turkey.

Aminbakhsh, S., & Sonmez, R. (2016). Discrete Particle Swarm Optimization Method for the Large-Scale Discrete Time-Cost Trade-off Problem. *Expert Systems with Applications*, *51*, 177–185.

Aminbakhsh, S., & Sonmez, R. (2017). Pareto Front Particle Swarm Optimizer for Discrete Time-Cost Trade-Off Problem. *Journal of Computing in Civil Engineering*, *31*(1), 04016040.

Anagnostopoulos, K. P., & Kotsikas, L. (2010). Experimental Evaluation of Simulated Annealing Algorithms for the Time-Cost Trade-off Problem. *Applied Mathematics and Computation*, *217*(1), 260-270.

Ashuri, B., & Tavakolan, M. (2012). Fuzzy Enabled Hybrid Genetic Algorithm–Particle Swarm Optimization Approach to Solve TCRO Problems in Construction Project Planning. *Journal of Construction Engineering and Management*, *138*(9), 1065-1074.

Benders, J.F. (1962). Partitioning Procedures for Solving Mixed Variables Programming Problems, *Numerische Mathematik*, 4, 238-252.

Bettemir, Ö. H. (2009). *Optimization of time-cost-resource trade-off problems in project scheduling using meta-heuristic algorithm*. Doctoral dissertation, Middle East Technical University, Ankara, Turkey.

Bettemir, Ö. H., & Birgönül, M. T. (2017). Network Analysis Algorithm for the Solution of Discrete Time-Cost Trade-off Problem. *KSCE Journal of Civil Engineering*, *21*(4), 1047–1058.

Bilir, M. (2015). *A Mixed Integer Programming Method for Pareto Front Optimization of Discrete Time Cost Trade-off Problem*. Master's thesis, Middle East Technical University, Ankara, Turkey.

Birbil, S., & Fang, S. C. (2003). An Electromagnetism-Like Mechanism for Global Optimization. *Journal of Global Optimization*, *25*,263–282

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.

Butcher, W.S. (1967). Dynamic Programming for Project Cost-Time Curves. *Journal of the Construction Division*, *Proceedings of the ASCE*, *93*, 59-73.

Chassiakos, A. P., & Sakellaropoulos, S. P. (2005). Time-Cost Optimization of Construction Projects with Generalized Activity Constraints. *Journal of Construction Engineering and Management-ASCE*, *131*(10), 1115-1124.

Chen, P. H., & Weng, H. (2009). A Two-Phase GA Model for Resource-Constrained Project Scheduling. *Automation in Construction*, *18*(4), 485–498.

Coello, C. A. C., Pulido, G. T., & Lechuga, M. S. (2004). Handling Multiple Objectives with Particle Swarm Optimization. *IEEE Transactions on Evolutionary Computation*, *8*(3), 256–279.

Colorni, A., Dorigo, M., & Maniezzo, V. (1992). Distributed Optimization by Ant Colonies. *Toward a Practice of Autonomous Systems*, 134-142.

Construction Industry Institute. (1988). *Concepts and Methods of Schedule Compression*. Austin, TX: Construction Industry Institute, University of Texas.

Crowston, W. B., & Thompson, G. L. (1967). Decision CPM: A Method for Simultaneous Planning, Scheduling and Control of Projects. *Operations Research*, *15*, 407-426.

De, P., Dunne, E. J., Ghosh, J. B., & Wells, C. E. (1995). The Discrete Time-Cost Tradeoff Problem Revisited. *European Journal of Operational Research*, *81*(2), 225-238.

De, P., Dunne, E. J., Ghosh, J. B., & Wells, C. E. (1997). Complexity of the Discrete Time-Cost Trade off Problem for Project Networks. *Operations Research*, *45*(2), 302–306.

Deckro, R. F., Hebert, J. E., Verdini, W. A., Grimsrud, P. H., & Venkateshwar, S. (1995). Nonlinear Time Cost Tradeoff Models in Project-Management. *Computers & Industrial Engineering*, *28*(2), 219-229.

Degirmenci, G., & Azizoglu, M. (2013). Branch and Bound Based Solution Algorithms for the Budget Constrained Discrete Time/Cost Trade-off Problem. *Journal of the Operational Research Society*, *64*(10), 1474–1484.

Demeulemeester, E. L., De Reyck, B., Foubert, B., Herroelen, W. S., & Vanhoucke, M. (1998). New Computational Results on the Discrete Time/Cost Trade-off Problem in Project Networks. *Journal of the Operational Research Society*, *49*(11), 1153-1163.

Demeulemeester, E. L., Dodin, B., & Herroelen, W. S. (1993). A Random Activity Network Generator. *Operations Research*, *41*(5), 972-980.

Demeulemeester, E. L., & Herroelen, W. S. (2002). *Project Scheduling: A Research Handbook*. Dordrecht, Boston: Kluwer Academic Publishers.

Demeulemeester, E. L., Herroelen, W. S., & Elmaghraby, S. E. (1996). Optimal Procedures for the Discrete Time Cost Trade-off Problem in Project Networks. *European Journal of Operational Research*, *88*(1), 50-68.

Demeulemeester, E., Vanhoucke, M., & Herroelen, W. (2003). RanGen: A Random Network Generator for Activity-On-The-Node Networks. *Journal of Scheduling*. *Vol. 6*, pp. 17–38.

Dragović, N., Vulević, T., Todosijević, M., Kostadinov, S., & Zlatić, M. (2017). Minimization of Direct Costs in the Construction of Torrent Control. *Technical Gazette*, *24*, 4(2017), 1123-1128.

Eberhart, R., & Kennedy, J. (1995). A New Optimizer Using Particle Swarm Theory. *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, 39-43.

El-Abbasy, M. S., Elazouni, A., & Zayed, T. (2016). MOSCOPEA: Multi-Objective Construction Scheduling Optimization Using Elitist Non-Dominated Sorting Genetic Algorithm. *Automation in Construction*, *71*(Part 2).

Elbeltagi, E., Hegazy, T., & Grierson, D. (2005). Comparison Among Five Evolutionary-Based Optimization Algorithms. *Advanced Engineering Informatics*, *19*(1), 43-53.

Elbeltagi, E., Hegazy, T., & Grierson, D. (2007). A Modified Shuffled Frog-Leaping Optimization Algorithm: Applications to Project Management. *Structure and Infrastructure Engineering*, *3*(1), 53-60.

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.

Eshtehardian, E., Afshar, A., & Abbasnia, R. (2008). Time–Cost Optimization: Using GA and Fuzzy Sets Theory for Uncertainties in Cost. *Construction Management and Economics*, *26*(7), 679-691.

Eshtehardian, E., Afshar, A., & Abbasnia, R. (2009). Fuzzy-based MOGA Approach to Stochastic Time-Cost Trade-off Problem. *Automation in Construction*, *18*(5), 692–701.

Eusuff, M. M., & Lansey, K. E. (2003). Optimizing of Water Distribution Network Design Using the Shuffled Frog Leaping Algorithm. *Journal of Water Resources Planning and Management*, *129*(3), 210–225

Falk, J. E., & Horowitz, J. L. (1972). Critical Path Problems with Concave Cost-Time Curves. *Management Science Series B-Application*, *19*(4), 446-455.

Fallah-Mehdipour, E., Bozorg Haddad, O., Rezapour Tabari, M. M., & Mariño, M. A. (2012). Extraction of Decision Alternatives in Construction Management Projects: Application and Adaptation of NSGA-II and MOPSO. *Expert Systems with Applications*, *39*(3), 2794–2803.

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.

Foldes, S., & Soumis, F. (1993). PERT and Crashing Revisited - Mathematical Generalizations. *European Journal of Operational Research*, *64*(2), 286-294.

Fondahl, J. M. (1961). A Non-Computer Approach to the Critical Path Method for the Construction Industry. *Technical Report, No. 9*, Construction Institute, Department of Civil Engineering, Stanford University, California.

Frank, H., Frisch, I. T., Van Slyke, R. & Chou, W. S. (1971). Optimal Design of Centralized Computer Networks. *Networks*, *1*, 43-57.

Fulkerson, D. R. (1961). A Network Flow Computation for Project Cost Curves. *Management Science*, *7*(2), 167-178.

Geem, Z. (2010). Multiobjective Optimization of Time-Cost Trade-off Using Harmony Search. *Journal of Construction Engineering and Management*, *136*(6), 711–716.

Goldberg, D. E., & Segrest, P. (1987). Finite Markov Chain Analysis of Genetic Algorithms. *Proceedings of the Second International Conference on Genetic Algorithms and their application*, Cambridge, Massachusetts, United States.

Goyal, S. K. (1975). A Note on A Simple CPM Time-Cost Trade-off Algorithm. *Management Science*, *21*(6), 718–722.

Hansen, M. P., & Jaszkiewicz, A. (1998). Evaluating the Quality of Approximations to the Non-Dominated Set. *Technical report IMM-REP1998-7*, Technical University of Denmark.

Hazır, Ö., Erel E., & Gunalay, Y. (2011). Robust Optimization Models for the Discrete Time/Cost Trade-off Problem. *International Journal of Production Economics*, *130*, 87-95.

Hazır, Ö., Haouari, M., & Erel, E. (2010). Discrete Time/Cost Trade-off Problem: A Decomposition-based Solution Algorithm for the Budget Version. *Computers & Operations Research*, *37*(4), 649–655.

Hegazy, T. (1999). Optimization of Construction Time-Cost Trade-off Analysis Using Genetic Algorithms. *Canadian Journal of Civil Engineering*, *26*(6), 685-697.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. Cambridge, MA, USA: University of Michigan Press.

Huang, Y., Zou, X., & Zhang, L. (2016). Genetic Algorithm – Based Method for the Deadline Problem in Repetitive Construction Projects Considering Soft Logic. *Journal of Management in Engineering*, *32*(4), 1–9.

Kalhor, E., Khanzadi, M., Eshtehardian, E., & Afshar, A. (2011). Stochastic Time-Cost Optimization Using Non-Dominated Archiving Ant Colony Approach. *Automation in Construction*, *20*(8), 1193–1203.

Kandil, A. (2005). *Multi-Objective Optimization for Large-Scale Highway Construction Projects*. Doctoral dissertation, University of Illinois, Urbana Champaign, Urbana, IL.

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.

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

Kelley, J. E., & Walker, M. R. (1959). Critical-Path Planning and Scheduling. *Proceedings of Eastern Joint Computer Conference*, *Vol. 16*, 160–173.

Kennedy, J., & Eberhart, R. C. (1995). Particle Swarm Optimization. *IEEE International Conference on Neural Networks Proceedings*, *Vols 1-6*, 1942-1948.

Kennedy, J., & Eberhart, R. C. (1997). A Discrete Binary Version of the Particle Swarm Algorithm. *Smc '97 Conference Proceedings - IEEE International Conference on Systems, Man, and Cybernetics*, *Vols 1-5*, 4104-4108.

Kerzner, H. (2009). *Project Management: A Systems Approach to Planning, Scheduling, and Controlling*. John Wiley & Sons.

Kirkpatrick, S., Gellat, C. D., & Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science*, *220*, 671 – 680.

Knowles, J., & Corne, D. (2002). On Metrics for Comparing Nondominated Sets. *IEEE Computer Society - Proceedings of the 2002 Congress on Evolutionary Computation, CEC 2002*, *Vol. 1*, pp. 711–716.

Kolisch, R., & Hartmann, S. (2006). Experimental Investigation of Heuristics for Resource-Constrained Project Scheduling: An Update. *European Journal of Operational Research*, *174*(1), 23–37.

Kolisch, R., Sprecher, A., & Drexl, A. (1995). Characterization and Generation of a General Class of Resource-Constrained Project Scheduling Problems. *Management Science*, *41*, 1693–1703.

Koo, C., Hong, T., & Kim, S. (2015). An Integrated Multi-Objective Optimization Model for Solving the Construction Time-Cost Trade-off Problem. *Journal of Civil Engineering and Management*, *21*(3), 323–333.

Land, A. H., & Doig, A. G. (1960). An Automatic Method of Solving Discrete Programming Problems. *Econometrica: Journal of the Econometric Society*, 497-520.

Li, H., & Love, P. (1997). Using Improved Genetic Algorithms to Facilitate Time-Cost Optimization. *Journal of Construction Engineering and management*, *123*(3), 233-237.

Liberatore, M., Pollack-Johnson, B., & Smith, C. (2001). Project Management in Construction: Software Use and Research Directions. *Journal of Construction Engineering and Management*, *127*(2), 101–107.

Liu, L., Burns, S. A., & Feng, C. W. (1995). Construction Time-Cost Trade-off Analysis Using LP/IP Hybrid Method. *Journal of Construction Engineering and Management*, *121*(4), 446-464.

Lock, D. (2007). *Project Management: 9Th Edition*. Surrey, UK: Gower Publication.

Menesi, W., Golzarpoor, B., & Hegazy, T. (2013). Fast and Near-Optimum Schedule Optimization for Large-Scale Projects. *Journal of Construction Engineering and Management*, *139*(9), 1117–1124.

Meyer, W. L., & Shaffer, L. R. (1963). Extensions of the Critical Path Method through the Application of Integer Programming. *Civil Engineering Construction Research Series No: 2*. University of Illinois, Urbana Champaign, Urbana, IL.

Meyer, W. L., & Shaffer, L. R. (1965). Extending CPM for Multiform Project Time-Cost Curves. *Journal of Construction Division*, *ASCE*, *91*(1), 45-68.

Mittelmann, H. D. (2013). Benchmarks for Optimization Software. Retrieved March 2014, from http://plato.asu.edu/bench.html

Moder, J. J., Phillips, C. R., & Davis, E. W. (1983). *Project Management with CPM, PERT, and Precedence Diagramming (3rd Ed.)*. New York: Van Nostrand Reinhold.

Monghasemi, S., Nikoo, M. R., Khaksar Fasaee, M. A., & Adamowski, J. (2015). A Novel Multi Criteria Decision Making Model for Optimizing Time-Cost-Quality Trade-off Problems in Construction Projects. *Expert Systems with Applications*, *42*(6), 3089–3104.

Moselhi, O. (1993). Schedule Compression Using the Direct Stiffness Method, *Canadian Journal of Civil Engineering*, *20*, 65-72.

Moussourakis, J., & Haksever, C. (2004). Flexible Model for Time/Cost Tradeoff Problem. *Journal of Construction Engineering and Management*, *ASCE*, *130*(3), 307-314.

Mubarak, S. (2010). *Construction Project Scheduling and Control*. John Wiley & Sons.

Mungle, S., Benyoucef, L., Son, Y. J., & Tiwari, M. K. (2013). A Fuzzy Clustering-based Genetic Algorithm Approach for Time-Cost-Quality Trade-off Problems: A Case Study of Highway Construction Project. *Engineering Applications of Artificial Intelligence*, *26*(8), 1953–1966.

Ng, S. T., & Zhang, Y. S. (2008). Optimizing Construction Time and Cost Using Ant Colony Optimization Approach. *Journal of Construction Engineering and Management*, *ASCE*, *134*(9), 721-728.

Okabe, T., Jin, Y., & Sendhoff, B. (2003). A Critical Survey of Performance Indices for Multi-Objective Optimisation. *The 2003 congress on evolutionary computation*, *CEC '03*, *IEEE Press*, *vol. 2*, pp. 878–85.

Panagiotakopoulos, D. (1977). A CPM Time-Cost Computational Algorithm for Arbitrary Activity Cost Functions. *INFOR: Information Systems and Operational Research*, *15*(2), 183-195.

Riquelme, N., Von Lucken, C., & Baran, B. (2015). Performance Metrics in Multi-Objective Optimization. *2015 Latin American Computing Conference*, *CLEI*, *IEEE*, pp. 1–11.

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

Rothfarb, B., Frank, H., Rosebaum, D., Steiglitz, K., & Kleitman, D. (1970). Optimal Design of Offshore Natural Gas Pipeline Systems. *Operations Research*, *18*, 992-1020.

Schwindt, C. (1995). *ProGen/Max: A New Problem Generator for Different Resource-Constrained Project Scheduling Problems with Minimal and Maximal Time Lags*. Research Report, WIOR 449, University of Karlsruhe.

Siemens, N. (1971). A Simple CPM Time-Cost Tradeoff Algorithm. *Management Science*, *17*(6), B354-B363.

Skutella, M. (1998). Approximation Algorithms for the Discrete Time-Cost Tradeoff Problem. *Mathematics of Operations Research*, *23*(4), 909-929.

Sonmez, R., & Bettemir, O. H. (2012). A Hybrid Genetic Algorithm for the Discrete Time-Cost Trade-off Problem. *Expert Systems with Applications*, *39*(13), 11428-11434.

Su, Z., Qi, J., & Wei, H. (2017). Simplifying the Nonlinear Continuous Time-Cost Tradeoff Problem. *Journal of Systems Science and Complexity*, *30*(4), 901–920.

Szmerekovsky, J.G., & Venkateshan, P. (2012). An Integer Programming Formulation for the Project Scheduling Problem with Irregular Time-Cost Tradeoffs. *Computers & Operations Research*, *39*, 1402-1410.

Tavares, L. V. (1999). *Advanced Models for Project Management*. Boston, MA: Kluwer Academic Publishers.

Vanhoucke, M. (2005). New Computational Results for the Discrete Time/Cost Trade-off Problem with Time-Switch Constraints. *European Journal of Operational Research*, *165*(2), 359-374.

Vanhoucke, M. (2015). *Generalized Discrete Time-Cost Tradeoff Problems*. In: Schwindt C, Zimmermann J (eds) Handbook on project management and scheduling, vol 1, pp 639–658, Berlin: Springer.

Vanhoucke, M., Coelho, J., Debels, D., Maenhout, B., & Tavares, L. V. (2008). An Evaluation of the Adequacy of Project Network Generators with Systematically Sampled Networks. *European Journal of Operational Research*, *187*(2), 511–524.

Vanhoucke, M., & Debels, D. (2007). The Discrete Time/Cost Trade-off Problem: Extensions and Heuristic Procedures. *Journal of Scheduling*, *10*(4-5), 311-326.

Vanhoucke, M., Demeulemeester, E., & Herroelen, W. (2002). Discrete Time/Cost Trade-offs in Project Scheduling with Time-Switch Constraints. *Journal of the Operational Research Society*, *53*(7), 741-751.

Veldhuizen, D. A. V. (1999). *Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations*. Doctoral dissertation, Graduate School of Engineering of the Air Force Institute of Technology, Air University, OH.

Xiong, Y., & Kuang, Y. P. (2008). Applying an Ant Colony Optimization Algorithm-based Multiobjective Approach for Time-Cost Trade-off. *Journal of Construction Engineering and Management*, *ASCE*, *134*(2), 153-156.

Yang, I. T. (2007a). Performing Complex Project Crashing Analysis with Aid of Particle Swarm Optimization Algorithm. *International Journal of Project Management*, *25*(6), 637-646.

Yang, I. T. (2007b). Using Elitist Particle Swarm Optimization to Facilitate Bicriterion Time-Cost Trade-off Analysis. *Journal of Construction Engineering and Management*, *ASCE*, *133*(7), 498-505.

Yang, H. H., & Chen, Y. L. (2000). Finding the Critical Path in An Activity Network with Time-Switch Constraints. *European Journal of Operational Research*, *120*(3), 603-613.

Zadeh, L. A. (1965). Fuzzy Sets. *Information and Control*, *8*(3), 338-353.

Zhang, H., & Li, H. (2010). Multi-Objective Particle Swarm Optimization for Construction Time-Cost Tradeoff Problems. *Construction Management and Economics*, *28*(1), 75–88.

Zhang, H., & Xing, F. (2010). Fuzzy-Multi-Objective Particle Swarm Optimization for Time-Cost-Quality Tradeoff in Construction. *Automation in Construction*, *19*(8), 1067-1075.

Zhang, L., Zou, X., & Qi, J. (2015). A Trade-Off Between Time and Cost in Scheduling Repetitive Construction Projects. *Journal of Industrial and Management Optimization*, *11*(4), 1423-1434.

Zhang, Y., & Ng, S. (2012). An Ant Colony System Based Decision Support System for Construction Time-Cost Optimization. *Journal of Civil Engineering and Management*, *ASCE*, *18*(4), 580–589.

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*, *ASCE*, *130*(2), 168-176.

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*, *ASCE*, *131*(1), 81-91.

Zitzler, E., Deb, K., & Thiele, L. (2000). Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation*, *8*(2), 173–195.

Zitzler, E., Laumanns, M., & Thiele, L. (2001). *SPEA2: Improving the Strength Pareto Evolutionary Algorithm*. TIK-report 103, Lausanne, Switzerland: Swiss Federal Institute of Technology.

Zitzler, E., & Thiele, L. (1998). Multiobjective Optimization Using Evolutionary Algorithms – A Comparative Case Study. *Proceedings of 5th International Conference on Parallel Problem Solving from Nature*, *(PPSN-V)*, pp. 292-301.

Zitzler, E., & Thiele, L. (1999). Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *IEEE Transactions on Evolutionary Computation*, *3*(4), 257–271.

Zou, X., Fang, S. C., Huang, Y. S., & Zhang, L. H. (2016). Mixed-Integer Linear Programming Approach for Scheduling Repetitive Projects with Time-Cost Trade-Off Consideration. *Journal of Computing in Civil Engineering*, *31*(3), 6016003.

CURRICULUM VITAE

Saman Aminbakhsh
Date of Birth: 25 Jan 1986
Place of Birth: Tabriz, Iran
E-mail: saman.aminbakhsh@metu.edu.tr
Cell-Phone: +90 536 761 72 61

EDUCATION

2013-2018	**Middle East Technical University (METU)**, Ankara, Turkey Doctor of Philosophy (Ph.D.), Civil Engineering Construction Engineering and Management CGPA: 4/4
2010-2013	**Middle East Technical University (METU)**, Ankara, Turkey Master's degree (M.Sc.), Civil Engineering Construction Engineering and Management CGPA: 4/4
2004-2009	**Islamic Azad University of Tabriz (IAUT)**, Tabriz, Iran Bachelor's degree (B.Sc.), Civil Engineering CGPA: 16.47/20

1997-2004 **National Organization for Development of Exceptional Talents (NODET)**, Tabriz, Iran

EXPERIENCE

2013-2018 **Teaching Assistant**
Civil Engineering Department, Construction Engineering and Management Division, Middle East Technical University (METU), Ankara, Turkey

2014-2016 **Research Assistant**
Civil Engineering Department, Construction Engineering and Management Division, Middle East Technical University (METU), Ankara, Turkey

PUBLICATIONS

A. SCI-E and SSCI Indexed Journal Articles:

A1. Aminbakhsh, S., Gunduz, M., & Sonmez, R. (2013). Safety risk assessment using analytic hierarchy process (AHP) during planning and budgeting of construction projects. *Journal of Safety Research*, 46, 99-105.

A2. Aminbakhsh S. & Sonmez R. (2016). Discrete Particle Swarm Optimization Method for the Large-Scale Discrete Time-Cost Trade-Off Problem. *Expert Systems with Applications*, 51, 177–185.

A3. Aminbakhsh S. & Sonmez R. (2017). A Pareto Front Particle Swarm Optimizer for Discrete Time-Cost Trade-Off Problem. *Journal of Computing in Civil Engineering*, ASCE, 31 (1), 04016040-1- 04016040-10.

A4. A Pareto Front Optimization Heuristic for the Large-Scale discrete Time-Cost Trade-Off Problem. (**Under Preparation**)

A5. Optimization of Large-Scale Discrete Time-Cost Trade-Off Problem using Mixed-İnteger Linear Programming. (**Under Preparation**)

B. International Conference Proceedings:

B1. Aminbakhsh S. & Sonmez R. (2015). Pareto Oriented Optimization of Discrete Time-Cost Trade-Off Problem Using Partıcle Swarm Optimization. *Procs 31th Annual ARCOM Conference*, Lincoln, UK, 33-40.

B2. Aminbakhsh S., Sonmez R. & Atan T. (2017). An Efficient Heuristic Method for Pareto Oriented Optimization of Time-Cost Trade-Off Problem for Construction Projects. *Procs. IO2017 The XVIII Congress of the Portuguese Association of Operational Research*, Valença, Portugal.

B3. Aminbakhsh S. & Sönmez R. (2018). Integrating a Meta-Heuristic Method into Microsoft Project for Time-Cost Trade-off Analyses. *5th İnternational Project and Construction Management Conference (IPCMC)*, Kyrenia, Cyprus. (**Abstract Accepted**)

C. National Conference Proceedings:

C1. Haghgooei A. & Aminbakhsh S. (2012). Türk Müteahhitlerin İran İnşaat Sektöründe Karşılaşabilecekleri Risklerin İncelenmesi. *2. Proje ve Yapım Yönetimi Ulusal Kongresi*, İzmir, Turkey.

C2. Aminbakhsh S, Sönmez R., Iranagh, M.A., & Rezvankhah E. (2014). Kuş Sürüsü Optimizasyon Algoritması ile Kesikli Zaman-Maliyet Ödünleşim Probleminin Çözümü. *3. Proje ve Yapım Yönetimi Ulusal Kongresi*, Antalya, Turkey.

C3. Aminbakhsh S., Sönmez R., & Bilir M. (2016). Tamsayılı Doğrusal Programlama Yöntemiyle Kesikli Zaman-Maliyet Ödünleşim Probleminin Optimal Pareto Çözümü. *4. Proje ve Yapım Yönetimi Ulusal Kongresi*, Eskişehir, Turkey.

D. Thesis:

D1. Aminbakhsh, S. (2013). *Hybrid Particle Swarm Optimization Algorithm for Obtaining Pareto Front of Discrete Time–Cost Trade-off Problem*. Master's thesis, Middle East Technical University, Ankara, Turkey.

E. Citing Articles (Web of Science, May 2018):

A1: 43

A2: 9

A3: 3

Total Citations: 55

AWARDS AND HONORS

2016	Awarded the Highest Cited Research certificate from Elsevier for an article published in Journal of Safety Research.
2001	Selected for National Organization for Development of Exceptional Talents (NODET) among over 800,000 participants.

TECHNICAL SKILLS

Adobe Photoshop
AIMMS-Advanced Optimization Software
Autodesk Sketchbook
C++/C# Programming
Microsoft Office
Microsoft Project
Microsoft Visio
Microsoft Visual Studio
Primavera PERTMaster-Risk analysis software
Primavera Project Management-P3 and P6
SPSS Statistics

LANGUAGES

Azeri	Mother Tongue
English	Fluent
Persian	Native Language
Turkish	Fluent

REFERENCES

References available upon request.