A two-phase GA model for resource-constrained project scheduling

Weng, Haijie

2005

Weng, H. (2005). A two-phase GA model for resource-constrained project scheduling.
Master's thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/46732>

<https://doi.org/10.32657/10356/46732>

Nanyang Technological University

Downloaded on 26 Apr 2023 19:46:53 SGT

a358653

**A TWO-PHASE GA MODEL
FOR
RESOURCE-CONSTRAINED PROJECT SCHEDULING**

WENG HAIJIE

**SCHOOL OF CIVIL AND ENVIRONMENTAL ENGINEERING
NANYANG TECHNOLOGICAL UNIVERSITY**

2005

HD
G9
.T54
W474
2005

**A TWO-PHASE GA MODEL
FOR
RESOURCE-CONSTRAINED PROJECT SCHEDULING**

WENG HAIJIE

SCHOOL OF CIVIL AND ENVIRONMENTAL ENGINEERING

**A thesis submitted to the Nanyang Technological University
in fulfilment of the requirements for the degree of
Master of Engineering**

2005

Acknowledgements

ACKNOWLEDGEMENTS

This report is the culmination of two years of effort. It has been individual work, but it certainly has not been solitary work. I am deeply indebted to the many people that have supported me in my research.

First and foremost, I would like to thank my supervisor, Dr. Chen Po Han, for his guidance and counsel without which my efforts would not have borne fruit. His broad range of knowledge and enthusiasm in pursuing various diverse interests will be a perennial source of inspiration.

My gratitude must be expressed to Dr. Yang Qizhen, Dr. Cui Lu, Mr. Zheng Daihua and Miss Wan Caiyun, for their invaluable suggestions on my research work.

I would also like to thank all my friends here at Nanyang Technological University for their constant support and for making my stay at Nanyang Technological University a truly memorable experience.

Thanks to my parents and girlfriend for their unconditional love and support. It is their dedication, sacrifice and unstinted faith in me that has made my pursuit of higher education possible.

Abstract

ABSTRACT

Project scheduling has a tremendous effect on the performance of a construction project. In project scheduling, problems can arise when each activity can be started at different time points and the resources needed by the activities are limited. Moreover, activities have required conditions to be met, such as precedence relationships, resource demands, etc. To resolve these problems, a two-phase GA (genetic algorithms) model was proposed in this research, in which both the effects of time-cost trade-off and resource scheduling were taken into account. A GA-based time-cost trade-off analysis was adopted to select the execution mode of each activity through the balance of time and cost, followed by utilization of a GA-based resource scheduling method to generate a feasible schedule which may satisfy all the project constraints.

The objective of this research is to integrate the ideas from the time-cost trade-off analyses and resource scheduling to form a new approach using a two-phase genetic algorithms (GAs) to tackle resource-constrained project scheduling (RCPS) problems and to develop a software system for RCPS which incorporates the proposed two-phase genetic algorithms using an object-oriented modeling approach.

An object-oriented Java application for RCPS was developed in this research. The application was developed in four main stages: requirement, analysis, design and implementation. These stages were designed to provide an effective model for RCPS and to satisfy the practical requirement in RCPS including (1) creation of a project plan; (2) optimization of schedule through time-cost analysis; (3) application of resource scheduling to reflect the real construction practice. The object-oriented model is finally mapped to Java code to produce a window-based application that runs on Microsoft XP with user-friendly interfaces. The demonstration showed the application works as expected.

Table of Contents

TABLE OF CONTENTS

<i>ACKNOWLEDGEMENTS</i>	<i>i</i>
<i>ABSTRACT</i>	<i>ii</i>
<i>TABLE OF CONTENTS</i>	<i>iii</i>
<i>LIST OF TABLES</i>	<i>vi</i>
<i>LIST OF FIGURES</i>	<i>vii</i>
<i>CHAPTER 1: INTRODUCTION</i>	<i>1</i>
1.1 Background	1
1.2 Research Objectives	5
1.3 Research Methodology	5
1.4 Organization of the Thesis	6
<i>CHAPTER 2: LITERATURE REVIEW</i>	<i>8</i>
2.1 Introduction	8
2.2 Characterization of the Resource-Constrained Project Scheduling Problems	8
2.3 Time-Cost Relationships in Construction Projects	10
2.4 Optimal Methods	11
2.4.1 Critical Path Method (CPM)	12
2.4.2 Mathematical Programming Methods	12
2.5 Heuristic Methods	15
2.6 Genetic Algorithms	17
2.7 Object-Oriented Modeling	19
2.8 Summary	19
<i>CHAPTER 3: GENETIC ALGORITHMS</i>	<i>21</i>
3.1 Introduction	21
3.2 Basic Concepts	21

Table of Contents

3.3 GAs for Time-Cost Trade-Off Problems	24
3.3.1 Basic Concepts and Terminology of Multiobjective Optimization	25
3.3.2 Traditional Approaches to Multiobjective Optimization Problems	29
3.3.3 Multiobjective Genetic Algorithms: A Genetic Approach to MOPs	29
3.3.3.1 Fitness Assignment and Selection	30
3.3.3.1.1 Aggregation Selection with Parameter Variation	30
3.3.3.1.2 Pareto Based Selection	31
3.3.3.2 Population Diversity	31
3.3.3.3 Elitism	32
3.4 GAs for Resource Scheduling Problems	32
3.4.1 Resource Scheduling in Construction	33
3.4.2 Problem Statement of Resource Scheduling	33
3.4.3 Genetic Approaches to Resource Scheduling Problems	35
3.5 Summary	36
CHAPTER 4: A TWO-PHASE GA MODEL FOR RESOURCE-CONSTRAINED PROJECT SCHEDULING	37
4.1 Introduction	37
4.2 Characteristics of Resource-Constrained Project Scheduling Problems	37
4.2.1 Activities	38
4.2.2 Resources	41
4.3 A Two-Phase GA Model for Resource-Constrained Project Scheduling	42
4.3.1 Chromosome Representation for Resource-Constrained Project Scheduling Problems	42
4.3.1.1 Choice of Decision Variable	43
4.3.1.2 Activity-to-Gene Mapping	43
4.3.1.3 Form of Coding	45
4.3.2 Proposed Two-Phase GA Model for Resource-Constrained Project Scheduling	45
4.3.2.1 Input Subsystem	46
4.3.2.2 Time-Cost Trade-Off Subsystem	48
4.3.2.3 Resource Scheduling Subsystem	54
4.3.2.3.1 Priority-Based Encoding	57
4.3.2.3.2 Procedures for Scheduling Resource-Constrained Activities	60
4.3.2.3.3 Flow of the Genetic Algorithms in Resource Scheduling Subsystem	64
4.3.2.4 Output Subsystem	67
4.4 Example Project	69
4.4.1 Planning Project with Precedence Relationships Only	71

Table of Contents

4.4.2 Adding Resource Capacities into Active Constraints	71
4.4.3 Allowing Interruption for Ongoing Activities and Overlap of Selected Activities	71
4.5 Summary	76
CHAPTER 5: JAVA APPLICATION DESIGN FOR RESOURCE-CONSTRAINED PROJECT SCHEDULING	78
5.1 Introduction	78
5.2 Requirements	78
5.3 Analysis	79
5.3.1 Requirement Analysis	79
5.3.2 Domain Analysis	81
5.3.2.1 Plan Package	82
5.3.2.2 GA Package	83
5.4 Design	86
5.4.1 Architecture Design	86
5.4.2 Detailed Design	88
5.4.2.1 Database Package	88
5.4.2.2 Business-Objects Package	89
5.4.2.3 User-Interface Package	91
5.5 Implementation	93
5.6 Demonstration	94
5.6.1 Menu Items	94
5.6.2 Gantt Chart	98
5.6.2.1 Precedence Relationships Only	99
5.6.2.2 Adding Resource Capacities	99
5.6.2.3 Adding All Project Constraints	100
5.7 Summary	101
CHAPTER 6: CONCLUSIONS	103
6.1 Report Review	103
6.2 Contributions	104
6.4 Future Research	105
REFERENCES	107
APPENDIX-JAVA SOURCE CODES	115

List of Tables

LIST OF TABLES

Table 4.1: Duration and resource consumption rate of each activity in the example 62

Table 4.2: Activity information 70

Table 4.3: GAs parameters value 70

List of Figures

LIST OF FIGURES

Figure 2.1: Optimal choice of a construction project.....	11
Figure 3.1: Chromosomal representation.....	22
Figure 3.2: Crossover.....	24
Figure 3.3: Mutation	24
Figure 3.4: Illustrative examples of Pareto optimality in objective space and possible relations of solutions in objective space	27
Figure 4.1: Example activity object	41
Figure 4.2: Network representation of a project in Directed Acyclic Graph (DAG).....	44
Figure 4.3: Topological Sorting for Figure 4.2	45
Figure 4.4: Operation of two-phase GAs model	47
Figure 4.5: Binary encoding.....	50
Figure 4.6: Crossover operator used in time-cost trade-off subsystem	50
Figure 4.7: Mutation operator used in time-cost trade-off subsystem.....	50
Figure 4.8: Trade-off curve and population ranking based upon non-dominated sorting	52
Figure 4.9: The flow of procedure in resource scheduling subsystem.....	56
Figure 4.10: Priority-based encoding	57
Figure 4.11: Network representation of a project in Directed Acyclic Graph (DAG) (Same as Fig 4.2).....	59
Figure 4.12: Activity priority in accordance with activities in Fig 4.11	59
Figure 4.13: Network representation of a project after activity 1 is removed	59
Figure 4.14: Topological sort of the Directed Acyclic Graph (DAG) in Fig 4.11	59
Figure 4.15: The flow chart for Step 3.....	63
Figure 4.16: The schedule of the example project.....	63
Figure 4.17: Resource profile of the example project	64
Figure 4.18: Crossover operator used in resource scheduling subsystem	65

List of Figures

Figure 4.19: Mutation operator used in resource scheduling subsystem.....	66
Figure 4.20: The final generation	68
Figure 4.21: The optimal tradeoff curve	68
Figure 4.22: The optimal choice	68
Figure 4.23: The network of example project	69
Figure 4.24: Final generation of the example project with precedence relationships only	72
Figure 4.25: Optimal trade-off curve of the example project with precedence relationships only .	72
Figure 4.26: Optimal choice of the example project with precedence relationships only	73
Figure 4.27: Schedule for duration of 49 days if planning project with precedence relationships only (hatched blocks stand for critical path)	73
Figure 4.28: Final generation of the example project under resource constraint	73
Figure 4.29: Optimal trade-off curve of the example project under resource constraint.....	74
Figure 4.30: Optimal choice of the example project under resource constraint	74
Figure 4.31: Schedule for duration of 56 days if adding resource capacities into active constraints (hatched blocks stand for critical path)	74
Figure 4.32: Final generation of the example project with all the constraints activated.....	75
Figure 4.33: Optimal trade-off curve of the example project with all the constraints activated	75
Figure 4.34: Optimal choice of the example project with all the constraints activated	75
Figure 4.35: Schedule for duration of 56 days if activating all the constraints (hatched blocks stand for critical path and solid blocks stand for completed part of the activities).....	76
Figure 5.1: Actor and use cases	81
Figure 5.2: Domain class structure for plan and schedule package.....	83
Figure 5.3: Domain class structure for GA package	85
Figure 5.4: Sequence diagram for the Add Crew scenario	85
Figure 5.5: Architectural overview of project scheduling system.....	88
Figure 5.6: Detailed design for "Plan" sub-package	90
Figure 5.7: Detailed design for "GA" sub-package	91
Figure 5.8: Class diagram in user-interface package	92

List of Figures

Figure 5.9: Sequence diagram for Add Task..... 93

Figure 5.10: Main window of the application 94

Figure 5.11: Menu items of the application 94

Figure 5.12: Saving project as an XML file..... 95

Figure 5.13: Project frame of the application..... 96

Figure 5.14: Task frame for creating new task 97

Figure 5.15: Task frame for viewing task property..... 97

Figure 5.16: Task notes of the application 97

Figure 5.17: GA frame of the application 98

Figure 5.18: Crew frame of the application 98

Figure 5.19: Gantt chart for optimal project duration of 49 days under precedence constraint 99

Figure 5.20: Gantt chart for optimal project duration of 56 days under precedence and resource
constraints..... 100

Figure 5.21: The Gantt chart at current stage before re-optimization..... 101

Figure 5.22: Gantt chart for optimal project duration of 56 days under all project constraints.... 101

CHAPTER 1 INTRODUCTION

1.1. Background

Construction industry projects involve complex packages of work for which the design and contracting organizations are responsible; the project is generally large, discrete and prototypical. Even on a small construction project, the number of possible ways of action and the number of ways to allocate resources quickly become overwhelming. These and other characteristics of the industry make particular demands upon the planning and scheduling techniques those have to be developed to serve it. Improper decisions on selecting construction methods and allocating resources, such as crew size and equipment, could lead to problems like cost overrun or project delay.

Conventional critical path method (CPM) techniques are widely used planning and scheduling tools in the construction industry. They assume in scheduling unlimited availability of resources for each activity. However, in practice, resources are available only in limited quantities and the resource demands of concurrent activities may not be satisfied. These techniques also assume non-interruption and non-overlap for construction activities. In real-world projects, an activity in construction may be interrupted, i.e., its resources may be assigned to other more important or urgent activities first, and then return to the activity itself. Normally, an activity cannot start until its predecessors have completed. However, overlap between an activity and its predecessor is possible in practice. For instance, an activity may start when its predecessor is partially, say 80%, completed. Furthermore, each activity could be performed in several ways depending on how resources needed are arranged and allocated. For example, if two excavators are assigned to an excavation work, it could be completed in half of the time required as it is done by one excavator. However, CPM techniques do not provide a method to decide which execution mode should be selected. Understanding of the abovementioned

Chapter 1. Introduction

drawbacks of CPM directed the research towards more general and practical resource-constrained project scheduling problems.

In resource-constrained project scheduling problems, each activity could be executed in more than one mode, and each mode might have different resource requirements, provided limited resource quantities. The general steps of handling resource-constrained project scheduling problems are as follows:

- Creation of a project plan. The plan refers to the proposal of different execution ways for each activity, the estimates of duration, cost and resource requirements for each execution mode, as well as the precedence relationships between activities. Moreover, other constraints, such as availability of resources, the interruption and/or overlap of activities, should be listed in the project plan.
- Generation of schedules. Because selection of different execution modes for each activity can result in different schedules, different feasible schedules are produced for the completion of a project within the project constraints. Schedules assign limited resources to activity at specific times. Resources include people, machines and raw materials.
- Evaluation of schedule performance. A set of measurements, such as total project duration, total project cost and resource utilization, are provided to measure the performance of different feasible schedules.
- Selection of an optimal schedule. The optimal schedule not only satisfies all the constraints, but also is optimal in given conditions, such as the shortest duration with fixed project cost, or the lowest project cost with fixed project duration.

The resource-constrained project scheduling problems (RCPSP) arise when each activity can be carried out in several possible ways. The trade-off between project direct cost and duration is the dominant consideration in selecting activity execution modes. For example, an execution mode which uses more productive equipment or hires more workers may give a shorter duration, but the corresponding cost may increase. Selecting a proper execution mode for each activity such that the project can be completed in the most cost-effective way within a specific completion time is desirable.

Chapter 1. Introduction

Time-cost trade-off analysis is to find the optimality of a schedule, i.e., minimizing the project cost while maintaining the desired project duration. However, the resource constraints determine the feasibility of a schedule and often affect the optimality of the schedule. For example, it may be easy to shorten a project's duration by assigning more resources to the tasks, but the availability of the limited resources may not satisfy the increased resource demands. Thus, allocation of resources is the key issue for generation of feasible schedules in resource-constrained project scheduling problems.

Planning and scheduling methods have been proposed and analyzed since at least 1950s. Although methods exist for finding optimal solutions to some specific scheduling problem formulations, many methods do not work when structure of the constraints or objectives change. For example, a scheduling heuristic that says "schedule the activities that project has shortest duration" may not perform well when the problem is modified to include limited availability of resources. In addition, many methods do not perform well or require a lot of computational effort once the number of options to complete an activity becomes too large or the network becomes too complex. In many cases, simply finding feasible solutions is a considerable challenge. The difficult nature of resource-constrained project scheduling makes project management and scheduling a challenge for operations research.

In general, scheduling problems are Non-Polynomial (NP)-hard, meaning there are no known algorithms for finding optimal solutions in polynomial time. Although some existing algorithms are able to find local optimal solutions, they take too long (computation time) when the problem size grows or when additional constraints are added. As a result, most research has been devoted to either simplifying the scheduling problem to the point where some algorithms can find solutions, or to devising efficient heuristics for finding near optimal good solutions.

Many solutions methods have been proposed and implemented. Early approach used mathematical programming techniques such as linear programming and dynamic programming to solve simplified versions of the problem exactly, but researchers later

Chapter 1. Introduction

realized that such methods can not solve the larger and more complex problems encountered in practice and are only suited for small-sized project. Heuristic methods were then devised to find good solutions, or to find simply feasible solutions for really difficult problems. They performed well over a variety of problems and widely used in practice because of their simplicity and ease of application. However, they proved to be very much problem dependent, with varying effectiveness on different cases. Furthermore, there was no way of knowing a priori the best set of heuristic rules to use for a given case.

The complex, combinatorial nature of most scheduling problems has led many researchers to experiment with genetic algorithms as a solution method. Genetic algorithms (GAs) are a stochastic search method introduced in the 1970s by Holland (Holland 1975). Based on simplifications of natural evolutionary processes, GAs operate on a population of solutions rather than a single solution and employ heuristics such as selection, crossover, and mutation to evolve better solutions. GAs do not rely much on assumptions or on heuristic rules, and they are robust. They are often noted for search large, multi-modal spaces effectively since they operate on a population of solutions rather than on one individual and do not use problem-dependent information.

Due to the continued challenge of resource-constrained project scheduling and the promising performance of genetic algorithms on similar problems, scheduling problems have attracted a great deal of attention in genetic algorithms.

To develop a flexible, powerful, maintainable and reusable software system for the resource-constrained project scheduling, an object-oriented approach that has gained increasing popularity is considered mostly suitable. The basic concept of object orientation is the object that abstracts a real world entity by encapsulating its characteristics (data and functionality). An object provides an interface for communication with other objects. Constructs such as inheritance and polymorphism allow easy extension and reusability of previously developed objects. The object-oriented

Chapter 1. Introduction

methodology provides an information processing paradigm for efficient development and management of complicated software system.

1.2 Research Objectives

The objective of this research is to integrate the ideas from the aforementioned research (time-cost trade-off analyses and resource scheduling) to form a new approach using two-phase genetic algorithms (GAs) to tackle resource-constrained project scheduling problems (RCPSP) and to develop a software system for resource-constrained project scheduling which incorporates the proposed two-phase genetic algorithms using an object-oriented modeling approach.

The specific objectives of this research are as follows:

- To develop a two-phase genetic algorithms for resource-constrained project scheduling problems;
- To build an object-oriented model which provides an object-oriented representation for modeling construction activities and incorporates project constraints and the two-phase genetic algorithms into the model;
- To implement the model in software system for solving real resource-constrained project scheduling problems.

1.3 Research Methodology

The research methodology has the following five major components:

- (1) Literature review
- (2) Algorithm development
- (3) Modeling
- (4) System building
- (5) Verification through testing

Chapter 1. Introduction

A two-phase genetic algorithms and an object-oriented model for resource-constrained project scheduling are developed through a survey of background literature on resource-constrained project scheduling, genetic algorithm-based time-cost optimization and resource scheduling, and object-oriented modeling. Using the developed object-oriented model and two-phase genetic algorithms model, a resource-constrained project scheduling system will be built. Finally, the system will be tested and analyzed through case example to prove the two-phase genetic algorithms built in the system and verify the efficiency and practicability of the system.

Five specific tasks are identified corresponding to the research methodology described above:

(1) Build up research background

The research built on three research foundations: (a) resource-constrained project scheduling, (b) genetic algorithm based cost-time optimization and resource scheduling, (c) object-oriented modeling and programming.

(2) Develop a two-phase genetic algorithms for resource-constrained project scheduling

(3) Develop an object-oriented model which incorporates the proposed two-phase genetic algorithms model

(4) Develop and implement the software system

(5) Test and verify the software system

1.4 Organization of the Thesis

This thesis is organized in the following sequence:

- Chapter 1 introduces the research background and confines the research objectives and research methodology.
- Chapter 2 outlines a comprehensive literature review which is relevant to this research.
- Chapter 3 describes the genetic algorithms relevant to this research, providing a background for the new genetic algorithms model for resource-constrained time-cost optimization problems

Chapter 1. Introduction

- Chapter 4 presents the two-phase genetic algorithms model for resource-constrained project scheduling problems.
- Chapter 5 describes the design of Java application for resource-constrained project scheduling.
- Chapter 6 offers the conclusions, contributions and recommendations for future work.

CHAPTER 2 LITERATURE REVIEW

2.1 Introduction

This chapter describes and evaluates the literature pertaining to this research. The characterization of resource-constrained project scheduling problem was reviewed, from where time-cost trade-off analyses and resource scheduling were identified as key issues to solve resource-constrained project scheduling problems. The relationships between project costs (direct and indirect) and project time were pointed out. Then research preciously accomplished in the area of resource-constrained project scheduling, emphasizing on time-cost trade-off modeling and resource scheduling, were described and areas of opportunity for new research were identified. Object-oriented modeling techniques were reviewed for the development of an object-oriented model for resource-constrained project scheduling.

2.2 Characterization of the Resource-Constrained Project Scheduling Problems

In general, a resource-constrained scheduling problem is defined as follows: A project consists of a set of interrelated activities. Each activity could be executed in more than one way, which can be represented using a continuous duration-resource function. Each activity could also be interrupted and/or overlapped. Resources are limited in quantity. Under these conditions, a solution which could find the optimal execution mode for each activity and properly assign the resources to the activities is to be researched to satisfy all the required constraints and produce the best time-cost combination.

Chapter 2. Literature Review

After sixty years of research directed at solving resource-constrained project scheduling problems, many recent works have been devoted to characterizing the problems. The purpose of these explorations has been to understand the structure of resource-constrained project scheduling problems so that problems can be generated in order to test the many solution methods.

Sprecher (1994) presented a formal formulation of the single- and multi-mode project scheduling problem including the general constraints and resource requirements. Kolisch et al. (1992) have characterized many variations of the resource-constrained project scheduling problems. They defined a set of parameters such as “resource strength” and “network complexity” that characterize the resource constraints and the number of precedence relationships.

In resource-constrained project scheduling problems, two characteristics draw researchers’ most attentions. One character is the activities are multi-modal, which means that activity may have more than one mode of execution. Each execution mode has its own set of resource requirements and estimated duration. The other character is activities are under constraints, such as precedence relationships between activities, resource availability and interruption and/or overlap of activities.

The trade-off between project direct cost and duration is the dominant consideration in selecting activity execution mode. Different execution mode of an activity has different resource requirements, estimated duration and cost. For example, an execution mode which uses more productive equipment or hires more workers may give a shorter duration, but the corresponding cost may increase. Selecting a proper execution mode for each activity such that the project can be completed in the most cost-effective way within a specific completion time is a key point to solve problems caused by the multi-mode of the activities.

Besides selecting appropriate execution mode for each activity, the assignment of resources to activities at specific time and generation of a feasible schedule which satisfy

Chapter 2. Literature Review

all project constraints are needed to be handled in order to solve resource-constrained project scheduling problems. Resource-constrained scheduling techniques compare the resource available to the project and those required for the individual activity at anytime. Another time span is considered if there are not sufficient resources available. The search is stopped when a suitable span is found. This approach can solve the problems induced by resource constraints and schedule project to satisfy all constraints.

2.3 Time-Cost Relationships in Construction Projects

There are two main types of cost associated with a construction project. One is direct cost, which includes the cost to the contractor of the labor, plant, and materials to carry out the activity. The other is indirect cost, which generally consists of costs incurred in direct proportion to the length of time that the contract takes- for example, the wages of the site staff or the head office expenses. Indirect cost will tend to increase directly as the length of the contract increases.

Construction time and cost are intricately related, e.g., leading to an increase in labor and plant costs (i.e., direct costs) when project duration is compressed; and project overhead (indirect costs) increasing with the project duration.

To minimize the costs associated with schedule compression, construction planners are urged to examine the time-cost optimization of construction activities before a decision is made. The first step for time-cost optimization is to find the trade-off curve between the direct cost and the project duration. Subsequently, construction planners can determine the total cost by summing up the estimated indirect cost and direct cost from the trade-off curve. As shown in Figure 2.1, the optimal choice to perform the project would be the lowest total cost. By adjusting the time for the project schedule to coincide with the optimal time obtained from the combined graphs, it is possible to carry out the work in the optimal overall duration.

Chapter 2. Literature Review

Fig. 2.1 Optimal choice of a construction project

2.4 Optimal Methods

Variations of the resource-constrained project scheduling problems have been proposed, implemented, and evaluated for over sixty years. In the past, the existing solution methods form two distinct classes: optimal methods and heuristic methods. Optimal methods, such as critical path method (CPM) and mathematical programming methods, are guaranteed to find an optimal solution if it exists. Heuristic solutions provide good solutions, but do not guarantee optimality.

When resource-constrained project scheduling solutions were first proposed, optimal methods were used for solving the problem. Given a problem, optimal methods find the best solution **every time they are run**. However, because the resource-constrained project scheduling problem is NP (non-polynomial)-hard, for large projects the size of the problem may render optimal methods computationally impracticable. Moreover, as constraints were added, the difficulty of finding optimal solution increased, and simply finding a good and feasible solution become good enough.

Chapter 2. Literature Review

2.4.1 Critical Path Method (CPM)

The critical path method (CPM) was developed to find shortest time to complete a project given estimates of activity durations. Many successful applications have been reported (Paulson 1973, McCough 1982, Cohenca et al. 1989). In the mean time, many drawbacks have also been identified (Badiru 1991, Bent and Thuman 1989). For example, CPM considers only logical constraints during planning, which is not real world of a construction process. In practice, the logical constraint is only one of many conditions to determine where an activity can be scheduled for construction or not. Other constraints may include resource constraints. CPM treats the construction of an activity as a non-stop process. In practice, an activity's construction may be interrupted or overlapped. Research has been conducted by addressing these drawbacks in order to enhance CPM-based techniques. Enhanced CPM based scheduling methods were presented, incorporating resource capacity into the scheduling process (Matthews 1994, Nkasu 1994). Re-definition and re-calculation of floats were discussed at the time of incorporating resources into scheduling (Bowers 1995, Gong and Rowings 1995). However, CPM becomes impractical when facing with problems of significant size or large sets of constraints. Furthermore, CPM only considers single-mode activity, which is impractical in real project scheduling problems.

2.4.2 Mathematical Programming Methods

Many project scheduling problems can be formulated using mathematical programming methods, but only if significant simplifications of the project scheduling problems are made.

One mathematical programming method used to develop optimal project plans is Linear Programming (LP). There are three general types of equations needed in an LP model. The first type of equations includes those that describe the behavior of the system. This set also requires the definition of a set of decision variables over which the various possible solutions are to be considered. Constraint functions, the second type of equations

Chapter 2. Literature Review

in LP models, are needed to restrict decision variables to values that are realistic for a specific problem. The third type of equation, actually a single function, is called the objective function. The objective function identifies the costs associated with decision variables and the direction of the objective, either minimization or maximization.

Kelly (1961) formulated time-cost trade-off problem by assuming linear time-cost relationships between activities. Other approaches, such as those by Hendrickson and Au (1989) and Pagnoni (1990), also used linear programming as the tool to solve the time-cost trade-off problem. Unfortunately, these models rarely reflect the majority of real-life problems facing the project planner. In the time-cost trade-off model, costs are more correctly represented by discontinuous step functions.

In situation where the discrete nature of decision variables is important, another mathematical programming method, Integer-Programming (IP), is often preferred.

Integer Programming may be used to model a variety of project planning problems (Hackman and Leachman 1989). Some of these problems include resource allocation, resource leveling, and the time-cost trade-off problem. Examples of the conditions under which integer and binary (either zero or one) variables may be employed in an IP model include:

- **Measuring Units:** Since the number of crew resources are discrete, an integer variables will be required to identify how many crews are available during a given work period.
- **Mutually Exclusive Decisions:** For example if a project planner must decide which crew is to be assigned to a given task, then a binary variable may be used to identify which crews are allocated to the project during each time period.
- **Internal States:** There may be several types of system “states” that should be explicitly modeled. Some examples of these states answer the following questions: “Has project j started?” “Has project j finished?” “Has activity i of project j been interrupted?” States are also modeled with binary variables.

Chapter 2. Literature Review

- Precedence Constraints: A matrix of binary variables is often the most compact way to identify the precedence constraints among activities.

IP techniques were proposed to solve practical construction scheduling problem with an objective of minimizing project duration under limited resource availability (Brand et al. 1964, Pritsker et al. 1969). An IP formulation of the time-cost trade-off problem was first proposed by Meyer and Shaffer (1963).

There are several difficulties when modeling problems using IP (Dyer 1992). The first is that IP models are static. Models are typically constructed to find a complete solution for a complete set of constraints. Changing those models, possibly as much as several times per day, could result in a situation in which the model is constantly being updated, but never actually used. The second problem with IP modeling, is the difficulty in representing conflicting objectives. For example, managers may wish to minimize duration and cost simultaneously. While it may be possible to represent these conflicting objectives as constraints when compared to some meta-objective, the model will not allow violation of some constraints to achieve a better overall solution. Finally, the system of equations solved with IP must be convex. Equations that should be modeled as step or point functions cannot be easily modeled in IP formulations.

In addition, IP techniques depend on characteristics of the objective function (strictly integer or binary value) and specific constraint formulation (only single-mode activity). As many constraints are found in real scheduling problems, IP can not perform well. IP, like LP, can be used only for small size problems.

Another approach, called Dynamic Programming (DP), describes the application of LP or IP in a sequential fashion. Dynamic programming is an approach used to decompose problems into “stages” and combine the solutions from each stage into a complete solution for the original problem. Robinson (1975), Elmagraby (1993) and De et al. (1995) used dynamic programming to solve time-cost trade-off problem for networks. Ressel and Caselton (1988) presented a dynamic programming formulation to minimize the

Chapter 2. Literature Review

project duration. Adeli and Karim (1997) presented a dynamic optimization formulation for the construction project scheduling problem, with the goal of minimizing the direct construction cost.

There are three critiques of DP. First is that most scheduling problems cannot be partitioned into stages since decisions made at one stage, necessarily impact decisions at other stages. The more important problem associated with DP is that the solution method requires the consideration of all possible state variables at a given stage. The number of these variables rises at an exponential rate with the size of the problem (Ravindran et al. 1987). Thus, the DP approach is also likely to be NP-hard. Finally, DP, like LP and IP, only considers single-mode activity.

These mathematical programming methods, however, were difficult to create, and these methods required a great deal of computational effort. Furthermore, the efficiency of the algorithms in searching for solutions depends on making strong assumptions about the objective functions (e.g. integer values) and constraints (e.g., single-mode activity) employed in the model, which might depart from real-world situations. In addition, they could not solve larger and more complex problems encountered in practice and were only suitable for small-size projects, although they were able to find local optimal solutions. Moreover, mathematical programming methods generally focused on a single objective.

2.5 Heuristic Methods

Whereas optimal solution methods are guaranteed to find the optimal solution (if one exists), heuristic methods sometimes find optimal solutions, but more often find simply good and feasible solutions. Heuristic algorithms have no guarantee of optimality as do optimal methods, but heuristic methods typically require far less time and effort than optimal methods.

Heuristic methods are based on rules of thumb, which generally lack mathematical rigor. Heuristics in scheduling are often referred to as scheduling rules. The definition of these

Chapter 2. Literature Review

rules is often quite complex, and most are tailored for a specific type of problem with a very specific set of constraints and assumptions. For example, in resource scheduling problems, heuristics are developed to allow a process of choosing between activities that are competing for the use of a scarce resource. Heuristics may be deterministic – they end up with the same result every time – or they may be stochastic – each time they may produce a different result. They may execute one role at a time, or they may be capable of parallel decisions. For example, in case of resource scheduling problems, serial heuristic methods select an order of activities to be scheduled prior to considering the resource constraint to be evaluated while parallel heuristics reorder the set of activities to be scheduled as the resource constraints are evaluated.

In resource-constrained project scheduling problems, heuristics operate on a set of tasks and determine when each task should be executed, which is in the scope of resource scheduling. If a task may be executed in more than one execution mode, the heuristic must also determine which execution mode to use, which is in the scope of time-cost trade-off analysis.

Examples of heuristic approaches to time-cost optimization problems include Fondahl's method (Fondahl 1961), Prager's structural model (Prager 1963), Siemens' effective cost slope model (Siemens 1971), and Moselhi's structural stiffness method (Moselhi 1993).

Examples of heuristic approaches to constrained-resource scheduling problems include two-phase heuristic method (Bell and Han 1991), local search heuristic method (Sampson and Weiss 1993), stochastic scheduling method (Drexl and Gruenewald 1993).

Panwalker and Iskander (1977) surveyed a range of heuristics from simple priority rules to very complex scheduling rules. Davis and Patterson (1975) compared eight standard heuristics on a set of single-mode resource-constrained project scheduling problems. The results showed that heuristics did not perform well when the resources were tightly constrained.

Chapter 2. Literature Review

Heuristic methods performed well over a variety of problems and are widely used in practice because of their simplicity and ease of application. However, they proved to be only suitable for small-size project and very much problem dependent, with varying effectiveness on different cases. Furthermore, there was no way of knowing a priori the best set of heuristic rules to use for a given case. No heuristics are developed to solve multi-mode resource-constrained project scheduling problems those are more prevailing in construction project.

2.6 Genetic Algorithms

Genetic algorithms (GAs) are a set of tools based on natural selection and the mechanisms of population genetics developed by John Holland (Holland 1975). GAs employ a random yet directed search inspired by the process of natural evolution and the principles of “survival of the fittest” for locating the globally optimal solution. One of the earliest suggested uses of genetic algorithms for scheduling was made by Lawrence Davis (1985). He noted that the attractiveness of using a stochastic search method due to the size of the search space and suggested an indirect representation in which the genetic algorithms operated on a list which was then decoded to form the actual schedule.

Since Davis’ paper, numerous implementations have been suggested for the general resource-constrained project scheduling problems with different variations. Satyanarayana et al. (1993) used GAs for resource allocation in construction projects. Chan et al. (1996) used GAs for construction resource scheduling which included both resource allocation and resource leveling. Li and Love (1997) presented several improvements in GAs used in solving the time-cost trade-off type of optimization problem in construction scheduling, which they termed time-cost optimization. Feng et al. (1997) presents an algorithm based on the principles of GAs for construction time-cost trade-off optimization. Leu et al. (2001) proposed a GA-based fuzzy construction time-cost trade-off model, in which the effects of both uncertain activity duration and time-cost trade-off are taken into account. The time-cost trade-off and resource-constrained scheduling were studied separately by those researchers. No researchers presented a

Chapter 2. Literature Review

genetic algorithm that could solve multi-mode resource-constrained problems which is more realistic in the construction industry. Feng et al. (1997) suggested that resource-constrained time-cost optimization problems could be solved by either using a penalty function or by using a domain-specific GA operator, but no results were released.

Genetic algorithms have been used to evolve heuristics for scheduling. Davis (1991) advocated hybridizing GA with the existing problem-solving algorithms so that the domain expertise could be preserved. Many successful applications have been reported to solve resource-constrained problems by hybridizing GA with heuristic methods (Chan et al. 1996; Gen and Cheng 1997).

Initial results indicate that genetic algorithms may provide a less computationally expensive means of searching NP-complete solution spaces (DeJong et al. 1989). Initial testing of genetic algorithms to application domains that are “Fuzzy, Uncertain and Non-deterministic (FUN)” appears to provide better results than heuristics in simultaneously more than one objective criterion. An evaluation of single-project, multi-resource allocation problem also provided good or better results than heuristic solutions (Chan et al. 1996)

One of the reasons that genetic algorithms may be effective, if allowed to run through sufficient generations, is that GAs may avoid hill climbing behavior associated some searching routines. An example of this “behavior” was noted by (Chan et al. 1996) in that, for single project multi-resource allocation problems, the GAs delayed some critical activities that heuristic solutions would not have delayed to ultimately arrive at a better overall solution.

In general, GAs are particularly suited for optimization problems in construction scheduling because, among other things, they do not experience combinatorial explosion; they do not rely much on assumptions or on heuristic rules, and they are robust. These characteristics allow GAs to overcome the difficulties associated with the nature of optimization problems in construction scheduling where other methods have failed.

Chapter 2. Literature Review

2.7 Object-Oriented Modeling

Object-oriented modeling is a new way of problem solving for the abstraction problems that exist in the real world. Its fundamental construct is the object, which combines both data structure and behavior in single entity. This is in contrast to conventional programming where data structure and behavior are only loosely connected. The object model encompasses the principle of abstraction, encapsulation, modularity, hierarchy, typing, concurrency and persistence. Object-oriented modeling has been recognized for its benefits such as reusability, stability, reliability, faster design and programming, easier maintenance, etc (Coad and Yourdon 1991; Cohen and Levitt 1991; Rumbaugh et al. 1991).

The implementation of object-oriented methodology is widely applied in the computer integrated construction. However, most of this research is on developing standard project information models to support current engineering in the architecture, engineering and construction (AEC) industry (Karim and Adeli 1999).

2.8 Summary

After review of the literature with many of the techniques described in this chapter, it is clear that none of the current project scheduling methods have been able to adequately solve the general resource-constrained project scheduling problems facing project planners. When the majority of time taken by project planners is used to select execution modes for activities and schedule activities under project constraints in order to achieve best project performance, it is no wonder that tools supporting only resource scheduling or time-cost analysis functions have made little impact on the industry. A tool that integrates both time-cost trade-off analysis and resource scheduling is desired.

This research is dedicated to the development of a fast and reliable GA model by which project planners may evaluate sets of alternative project plans and produce optimal and

Chapter 2. Literature Review

feasible project schedules. An objected-oriented software system with the built-in GA model is developed to support project planners' work with user-friendly interfaces.

CHAPTER 3 GENETIC ALGORITHMS

3.1 Introduction

This chapter described the genetic algorithms relevant to this research, providing a background for the new genetic algorithms for resource-constrained time-cost optimization problems proposed in the next chapter. Firstly, the basic concepts of genetic algorithms are introduced. This is followed by the description and evaluation of genetic algorithms with time-cost trade-off and resource scheduling problems. Finally, a summary is given.

3.2 Basic Concepts

Genetic algorithms (GAs) are search algorithms developed by Holland (1975), which are based on the mechanics of natural selection and genetics to search through decision space for optimal solutions (Goldberg 1989). The metaphor underlying GAs is natural selection. In evolution, the problem each species faces is to search for beneficial adaptations to the complicated and changing environment. In other words, each species has to change its chromosome combination to survive in the living world.

Genetic algorithms, differing from conventional search techniques, start with an initial set of random solutions called population. Each individual in the population is called a chromosome, representing a solution to the problem. A chromosome is a string of symbols; it is usually a binary string, an integer string or a real number string. The chromosomes evolve through successive iterations, called generations. During each generation, the chromosomes are evaluated, using some measures of fitness. To create the next generation, new chromosomes, called offsprings, are formed by either (a) merging

Chapter 3. Genetic Algorithms

two chromosomes from the current generation using a crossover operator or (b) modifying a chromosome using a mutation operator. A new generation is formed by (a) selecting, according to the fitness values, some of the parents and offspring and (b) rejecting others so as to keep the population size constant. Fitter chromosomes have higher probabilities of being selected. After several generations, the algorithms converge to the best chromosome, which hopefully represents the optimum or sub-optimal solution to the problem. Let $P(t)$ and $C(t)$ be parents and offsprings in the current generation t ; a typical optimization procedure involving genetic algorithm is summarized as below:

```
Begin
t = 0;
Initialize P (t);  (Randomly generate a new population of solutions)
Evaluate P (t);  (Evaluate the new solutions)
While (terminating condition not met) do {
    Recombine P (t) to yield C (t);  (Recombine solutions using genetic operators)
    Evaluate C (t);
    Select P (t+1) from C (t);  (Select the better solutions)
    t = t + 1;}
End
```

As each individual solution is represented by a chromosome, the chromosome typically consists of a number of genes, which may be visualized as boxes arranged in a linear fashion (Figure 3.1). Two attributes are associated with each gene: position and content which codes for a solution.

Fig. 3.1 Chromosomal representation

Chapter 3. Genetic Algorithms

Usually, initialization is assumed to be random. After initialization, individuals generated are evaluated quantitatively by applying an objective function on the solution encoded by their chromosomal representation; this is a direct indication of worth of the solution in the task environment. Recombination typically involves crossover and mutation to yield offspring.

Crossover is the main genetic operator. It operates on two chromosomes at a time and generates offspring by combining both chromosomes' features. A simple way to achieve crossover would be to choose a random cut-point and generate the offspring by combining the segment of one parent to the left of the cut-point with the segment of the other parent to the right of the cut-point (as depicted in Figure 3.2). Although crossover is principally thought of as a mechanism that improves the quality of solution, it is also possible that crossover will disrupt a good chromosome already present, especially the long ones (Goldberg 1989).

Besides crossover, mutation is a background operator which produces spontaneous random changes in various chromosomes. A simple way to achieve mutation would be to alter one gene (as depicted in Figure 3.3). The principal use of mutation is to reintroduce genetic diversity to avoid getting trapped in local optima; the frequency of mutation is often kept very low in order to avoid disruption of good solutions.

Selection is to create new populations from generation to generation. Among the methods studied in literature, roulette wheel and ranking are widely used. Essentially, selection places the better chromosomes in an intermediate mating pool according to the fitness magnitude of each chromosome.

GAs are not random "generate and test" procedures, but incorporate a very effective and implicit learning element in the algorithm through selection and recombination. The principles of GAs have been well-described in Goldberg (1989) and Davis (1991).

Chapter 3. Genetic Algorithms

Fig. 3.2 Crossover

Fig. 3.3 Mutation

3.3 GAs for Time-Cost Trade-Off Problems

In order to solve the problems caused by the multi-mode of activities in resource-constrained project scheduling, time-cost trade-off analysis is essential to make decisions to choose the proper methods, resources, and equipments to perform each activity of a project, which optimizes the overall performance of the project in terms of time and cost. The process of finding the optimal time-cost trade-off curve of a project is similar to minimizing time and cost simultaneously – a multiobjective optimization. The following sections describe the concept of multiobjective optimization and GAs approach to multiobjective optimization problems.

Chapter 3. Genetic Algorithms

3.3.1 Basic Concepts and Terminology of Multiobjective Optimization

Many real-world decision making problems involve multiple objectives, which need to be optimized simultaneously. This kind of problem is the so-called multiobjective optimization problem (MOP). Multiobjective optimization problems (MOP) are common. For example, in the construction industry, the cost of a project is to be minimized, while a minimum duration is desired. Besides cost and duration, other objectives may be important such as resource consumption. They can be either defined explicitly as separate optimization criteria or formulated as constraints, e.g., the resource consumption rate of the project must not exceed the resource availability. Formally, this can be defined as follows:

Definition. 1: (Multiobjective Optimization Problem) A general MOP includes a set of n parameters (decision variables), a set of k objective functions, and a set of m constraints. Objective functions and constraints are functions of the decision variables. The optimization goal is to

$$\text{maximize } y = f(x) = (f_1(x), f_2(x), \dots, f_k(x))$$

$$\text{subject to } e(x) = (e_1(x), e_2(x), \dots, e_m(x)) \leq 0$$

$$\text{where } \begin{array}{l} x = (x_1, x_2, \dots, x_n) \in X \\ y = (y_1, y_2, \dots, y_k) \in Y \end{array}$$

and x is the decision vector, y is the objective vector, X is denoted as the decision space, and Y is called the objective space. The constraint $e(x) \leq 0$ determines the set of feasible solution.

Definition. 2 (Feasible Set) The feasible set X_f is defined as the set of decision vectors x that satisfy the constraints $e(x)$: $X_f = \{x \in X | e(x) \leq 0\}$

Chapter 3. Genetic Algorithms

Without loss of generality, a minimization problem is assumed here. For maximization or mixed maximization/minimization problems the definitions are similar to what presented in this section.

Consider again the above example in the construction industry and assume that the two objectives, cost (f_1) and duration (f_2), are to be minimized under a resource constraint (e_1). Then an optimal solution might achieve minimum duration at minimal cost and does not violate the resource limitations. If such a solution exists, we actually only have to solve a single objective optimization problem (SOP). The optimal solution for either objective is also the optimum for the other objective. However, what makes MOPs difficult is the common situation when the individual optima corresponding to the distinct objective functions are different. Then, the objectives are conflicting and cannot be optimized simultaneously. For example, what corresponding to the reduction in activity duration is an increase in cost. This is because that some ways by which crashing is achieved include overtime, additional manpower and/or equipment, and the use of better skilled men and/or improved technology, those can incur additional cost. Hence, a satisfactory trade-off has to be found. In a construction project, duration and cost are generally competing: crashed duration substantially increases cost, while normal duration usually provides lower cost. Depending on the owner's requirements, an intermediate solution (medium duration, medium cost) might be an appropriate trade-off. This discussion makes clear that a new notion of optimality is required for MOPs.

In single-objective optimization, the feasible set is completely (totally) ordered according to the objective function f : for two solutions $a, b \in X_f$ either $f(a) \geq f(b)$ or $f(b) \geq f(a)$. The goal is to find the solution that gives the maximum value of f (Cohon 1985). However, when several objectives are involved, the situation changes: X_f is, in general, not totally ordered, but partially ordered (Pareto 1896). This is illustrated in Figure 3.4 (a). The solution represented by point B is better than the solution represented by point C : it provides a shorter duration with a lower cost. It would be even preferable if it would only improve one objective, as is the case for C and D : despite equal cost, C achieves shorter

Chapter 3. Genetic Algorithms

duration than D . In order to express this situation mathematically, the relations “=”, “<” and “>” are extended to objective vectors by analogy to the single-objective case.

Fig. 3.4 Illustrative examples: (a) Pareto optimality in objective space; (b) possible relations of solutions in objective space.

Definition. 3: For any two objective vectors u and v ,

$$u = v \text{ iff } \forall i \in \{1, 2, \dots, k\} : u_i = v_i$$

$$u \geq v \text{ iff } \forall i \in \{1, 2, \dots, k\} : u_i \geq v_i$$

$$u > v \text{ iff } u \geq v \wedge u \neq v$$

Using this notion, it holds that $B > C$, $C > D$, and, as a consequence, $B > D$. However, when comparing B and E , neither can be said to be superior, since $B \not> E$ and $E \not> B$. Although the solution associated with E has shorter duration, it provides higher cost than the solution represented by B . Therefore, two decision vectors a , b can have three possibilities with MOPs regarding the $\geq$ relation (in contrast to two with SOPs): $f(a) \geq f(b)$, $f(b) \geq f(a)$, or $f(a) \not\geq f(b) \wedge f(b) \not\geq f(a)$. Here, the following symbols and terms are used in order to classify the different situations.

Definition. 4: (Pareto Dominance) For any two decision vectors a and b ,

Chapter 3. Genetic Algorithms

$a \succ b$ (a dominates b) iff $f(a) \succ f(b)$

$a \succeq b$ (a weakly dominates b) iff $f(a) \succeq f(b)$

$a \sim b$ (a is indifferent to b) iff $f(a) \not\succeq f(b) \wedge f(b) \not\succeq f(a)$

In Figure 3.4 (b), the north-east rectangle encapsulates the region in objective space that is dominated by the decision vector represented by B . The south-west rectangle contains the objective vectors whose corresponding decision vectors dominate the solution associated with B . All solutions for which the resulting objective vector is in neither rectangle are indifferent to the solution represented by B .

Based on the concept of Pareto Dominance, the optimality criterion for MOPs can be introduced. Still referring to Figure 3.4, A is unique among B , C , D , and E : its corresponding decision vector a is not dominated by any other decision vector. That means, a is optimal in the sense that it cannot be improved in any objective without causing a degradation in at least one other objective. Such solutions are denoted as *Pareto optimal*.

In Figure 3.4 the points on the dotted curve represent Pareto-optimal solutions. They are indifferent to each other. This makes the main difference between MOPs and SOPs clear: there is no single optimal solution but rather a set of optimal trade-offs. None of these can be identified as better than the others. The entirety of all Pareto-optimal solutions is called the Pareto optimal set; the corresponding objective vectors form the Pareto-optimal front.

Basically, we can treat construction time-cost trade-off problem as a multiobjective optimization problem, which tries to minimize both cost and project duration. Each member in the population has its own total project duration and cost; therefore, a nondominated set (a trade-off curve) can be determined such that there are no members in the population that have better objective values in both time and cost than the members in the nondominated set. When compressing activities to obtain the trade-off curve, resource-constrained problems may arise when the amount of required resource at any instant exceeds the available resource. Hence the resource availability will affect the

Chapter 3. Genetic Algorithms

determination of the project duration. The constraints to the MOPs will be discussed later in this chapter.

3.3.2 Traditional Approaches to Multiobjective Optimization Problems

As most optimization problems are multiobjective to their nature, there are many methods available to tackle these kinds of problems. Traditional methods for generating the Pareto-optimal set aggregate the objectives into a single, parameterized objective function by analogy to decision making before search. However, the parameters of this function are not set by the decision-maker, but systematically varied by the optimizer. Several optimization runs with different parameter settings are performed in order to achieve a set of solutions which approximates the Pareto-optimal set. Basically, this procedure is independent of the underlying optimization algorithm.

Some representatives of this class of techniques are the weighting method (Cohon 1978), the constraint method (Cohon 1978), goal programming (Steuer 1986), and the minmax approach (Koski 1984).

3.3.3 Multiobjective Genetic Algorithms: A Genetic Approach to MOPs

For effective construction time-cost optimization, a multiobjective optimization approach should be introduced to allow the algorithm within the multiobjective space greater freedom to explore the possible solutions, thereby reducing the likelihood of being trapped in the local optima (Knowles et al. 2001). Moreover, since there are numerous activities within a project, it is almost impossible to evaluate all possible combinations within a short period of time and at a reasonable cost (Ng et al. 2000). A searching tool is therefore indispensable for efficient and comprehensive time-cost optimization. Being regarded as a powerful technique for locating the global optimum, genetic algorithms have a high potential for fulfilling the role of a basic searching tool during the optimization process.

Chapter 3. Genetic Algorithms

Recently, genetic algorithms have become established as an alternative to traditional methods through which i) large search spaces can be handled and ii) multiple alternative trade-offs can be generated in a single optimization run. Lately there has been a large development of different types of multiobjective genetic algorithms. The big advantage of genetic algorithms over traditional methods is that a genetic algorithm manipulates a population of individuals. It is therefore tempting to develop a strategy in which the population captures the whole Pareto front in one single optimization run.

When applying genetic algorithm to multiobjective optimization problems, three major problems must be addressed:

1. How to accomplish fitness assignment and selection, respectively, in order to guide the search towards the Pareto-optimal set.
2. How to maintain a diverse population in order to prevent premature convergence and achieve a well distributed and well spread nondominated set.
3. How to prevent losing the best solutions due to sampling effects during the selection process.

In the following of this section, three key issues in multiobjective genetic algorithms and a categorization of general techniques which deal with these issues are presented.

3.3.3.1 Fitness Assignment and Selection

In contrast to single-objective optimization, where objective function and fitness function are often identical, both fitness assignment and selection must allow for several objectives with multiobjective optimization problems. The following are some widely adopted fitness assignment and selection methods.

3.3.3.1.1 Aggregation Selection with Parameter Variation

As with these methods, the objectives are aggregated into a single parameterized objective function; however, the parameters of this function are not changed for different

Chapter 3. Genetic Algorithms

optimization runs, but instead systematically varied during the same run. Some approaches (Hajela and Lin 1992, Ishibuchi and Murata 1996), for instance, use the weighting method. Since each individual is assessed using a particular weight combination (either encoded in the individual or chosen at random), all members of the population are evaluated by a different objective function. Hence, optimization is done in multiple directions simultaneously. Nevertheless, the potential disadvantages of the underlying scalarization method (e.g., a bias towards convex portions of the Pareto-optimal front) may restrict the effectiveness of such multiobjective genetic algorithms (Veldhuizen 1999).

3.3.3.1.2 Pareto Based Selection

The concept of calculating an individual's fitness on the basis of Pareto dominance was first suggested by Goldberg (1989). He presented a "revolutionary 10-line sketch" (Deb 1999) of an iterative ranking procedure: First all non-dominated individuals are assigned rank one and temporarily removed from the population. Then, the next nondominated individuals are assigned rank two and so forth. Finally, the rank of an individual determines its fitness value. Remarkable here is the fact that fitness is related to the whole population, while with other aggregation techniques an individual's raw fitness value is calculated independently of other individuals. This idea has been taken up by numerous researchers, resulting in several Pareto-based fitness assignment schemes (e.g., Fonseca and Fleming 1993, Horn et al. 1994, Srinivas and Deb 1994).

3.3.3.2 Population Diversity

In order to approximate the Pareto-optimal set in a single optimization run, evolutionary optimizers have to perform a multi-modal search where multiple, widely different solutions are to be found. Therefore, maintaining a diverse population is crucial for the efficacy of a multiobjective genetic algorithm. To achieve population diversity, several methods have been developed; the one most frequently used in evolutionary multiobjective optimization is fitness sharing.

Chapter 3. Genetic Algorithms

Fitness sharing (Goldberg and Richardson 1987), which is the most frequently used technique, aims at promoting the formulation and maintenance of stable subpopulations (niches). It is based on the idea that individuals in a particular niche have to share the available resources. The more individuals are located in the neighborhood of a certain individual, the more its fitness value is degraded.

The fitness sharing method is relatively complicated and difficult to be implemented. To keep population diverse, one simple way is to set a random selection probability; hence chromosomes can be randomly selected without being affected by their fitness value. Thus, the diversity of the population can be kept to some extent.

3.3.3.3 Elitism

De Jong (1975) suggested a policy to always include the best individual of P_t into P_{t+1} in order to prevent losing it due to sampling effects or operator disruption. This strategy, which can be extended to copy the b best solutions to the next generation, is denoted as elitism.

In general, by directly using De Jong's idea, one strategy to incorporate elitism into multiobjective genetic algorithms is to copy those individuals from P_t automatically to P_{t+1} whose encoded decision vectors are nondominated in that run of generation (Tamaki et al. 1994).

3.4 GAs for Resource Scheduling Problems

In the previous section, multiobjective optimization problems and genetic approaches to such kind of problems have been studied under the assumption that there are no constraints with them. However, optimization deals with problems of minimizing or maximizing single or multiple objectives with several variables usually subject to constraints. For example, in the multiobjective optimization problems discussed before, the project duration is subject to both precedence constraints and resource constraints.

Chapter 3. Genetic Algorithms

Within the scope of this research, the constrained optimization problem is confined to the resource scheduling problems in the construction industry.

3.4.1 Resource Scheduling in Construction

When crashing activities to obtain the trade-off curve to conduct time-cost optimization analysis, resource constrained problems may arise when the amount of required resources at any instant exceeds the available resource.

For the crashed activities, corresponding to the reduction in activity duration is an increase in resource consumption. Some ways by which crashing is achieved include overtime, additional manpower and/or equipment, and the use of better skilled men and/or improved technology. Hence the crashed activities are characterized by shorter duration and higher resource requirements comparing with normal activities. The resource scheduling problem is common to occur under the condition of the time-cost trade-off optimization.

Besides resource and precedence constraints, construction project is subject to other constraints, such as interruption of ongoing activities and overlap of activities. These constraints should be considered in resource scheduling in order to generate feasible project schedules.

3.4.2 Problem Statement of Resource Scheduling

The problem of scheduling activities under resource restrictions with respect to the objective of minimizing the project duration is referred to the literature as a resource scheduling problem. The basic type of the problem can be stated as follows:

- A project consists of a number of interrelated activities, each characterized by a known duration and given resource requirements.
- Start time of each activity is dependent upon the completion of some other activities (precedence constraints).

Chapter 3. Genetic Algorithms

- In construction practice, resources are available in limited quantities but are renewable from period to period.
- There is no substitution between resources.
- Activity precedence relationships may include overlap so that a given activity may begin when its predecessor is partially complete.
- Ongoing activities may be interrupted. An ongoing activity may be defined so that its resources may be applied to a different but urgent activity, and then returned to the original activity.

A solution is to determine the start times of activities with respect to all the project constraints so as to minimize the project duration.

There are two essential issues to be dealt with for the resource-constrained scheduling problems:

- Determine the order of activities without violating the precedence constraints;
- Determine the earliest start time for each activity according to available resources and interruption and/or overlap of activities.

The problem can be stated mathematically as follows (firstly we assume that duration is selected from the alternative durations for each activity):

$\min t_n$ [To obtain the earliest starting time of last activity]

subject to $t_j - t_i \geq d_i, \forall j \in S_i$ [Constraint 1]

$\sum_{i \in A_k} r_{ik} \leq b_k, k = 1, 2, \dots, m$ [Constraint 2]

$t_i \geq 0, i = 1, 2, \dots, n$

where t_i is the starting time of activity i , n is the total number of activities, d_i the duration of activity i , S_i the set of successors of activity i , r_{ik} the amount of resource k required by activity i , b_k the total availability of resource k , A_{t_i} the set of activities in process at time t_i , and m the number of different resource types. Constraint 1 ensures that none of the

Chapter 3. Genetic Algorithms

precedence constraints is violated. Constraint 2 ensures that the amount of resource k used by all activities does not exceed its quantity limit in any period.

3.4.3 Genetic Approaches to Resource Scheduling Problems

As reported in the literature, genetic algorithms have been used as a new approach to overcome some of the previously mentioned difficulties associated with analytical and heuristic methods. The GAs schedule the starts of activities in a single project and hence perform resource allocation with the objective of minimizing the project duration under limited resource constraints.

In applying GAs to the resource scheduling problems, the issues of the treatment of illegal schedules introduced by the GA operations often come up. Schedules consisting of ordered lists of activities often produced new schedules with activities duplicated and/or missing or violating precedence and resource constraints as a result of crossover. Many representational schemes have been proposed for solving such difficulties associated with GAs in applying in scheduling problems.

Generally, the representation schemes proposed for a scheduling problem can be classified into direct representation and indirect representation. In the indirect representation, genetic algorithms work on a population of encoded solutions, and a transition from chromosome representation to a legal production schedule has to be performed by a schedule builder prior to evaluation. For example, Syswerda (1991), who used GAs for scheduling laboratory activities with precedence and resource constraints, solved the problems by using a schedule builder (incorporating domain-specific heuristics) to repair the illegal schedules. This adds a considerable amount of work outside the main GAs and the efficiency of GAs search is reduced.

In the direct problem representation, the schedule itself is used as a chromosome, and no decoding procedure is required, but it takes a great deal of effort to develop complicated genetic operators.

Chapter 3. Genetic Algorithms

The resource scheduling problem can essentially be viewed as a kind of ordering problem subject to some constraints because once the order of activities is given, the possible earliest start time for each activity can be easily determined according to the available resources. Hence, the main concern is to find out an appropriate order of activities with GAs. An indirect problem representation is used in this research, which is discussed in the next chapter.

3.5 Summary

The chapter has identified several issues concerning solving resource-constrained project scheduling problems by using genetic algorithms. Analogous to natural selection and genetic in reproduction, genetic algorithms (GAs) have been successfully adopted to solve many science and engineering problems and have proven to be an efficient means for searching optimal solutions in a large problem domain. The resource-constrained project scheduling problem can be treated as a combination of multiple objective problem (time-cost trade-off) and resource scheduling problem. In this research, it is attempted to solve the resource-constrained project scheduling problem by integrating the two different genetic algorithms (multiobjective genetic algorithm and resource-constrained genetic algorithm). In brief, the genetic algorithms description and the computer system built on those algorithms in the following chapters were derived from the outcome of the genetic algorithms described and evaluated in this chapter.

CHAPTER 4 A TWO-PHASE GA MODEL FOR RESOURCE-CONSTRAINED PROJECT SCHEDULING

4.1 Introduction

This chapter describes the two-phase GA model for resource-constrained project scheduling in three parts: (1) characteristics of resource-constrained project scheduling problems, (2) the two-phase GA model, which consists of four subsystems: input subsystem, time-cost trade-off subsystem, resource-scheduling subsystem and output subsystem, and (3) example project. The characteristics of resource-constrained project scheduling problems are in the form of assumptions about activities and resources with their associated constraints. The variations of project scheduling problems that can be solved can be determined through the characterization of the resource-constrained project scheduling problems. A two-phase GA model is developed based on the characterization of the problems and demonstrated through an example project.

4.2 Characteristics of Resource-Constrained Project Scheduling Problems

In general, a resource-constrained scheduling problem is defined as follows: A project consists of a set of interrelated activities. Each activity could be executed in more than one way, which can be represented using a continuous duration-resource function. Each activity could also be interrupted and/or overlapped. Resources are limited in quantity. Under these conditions, a solution which could find the optimal execution mode for each activity and properly assign the resources to the activities is to be researched to satisfy all the required constraints and produce the best time-cost combination. The following

Chapter 4. A Two-Phase GA Model for Resource-Constrained Project Scheduling

sections list the characteristics of resource-constrained project scheduling problems from two perspectives: activities and resources.

4.2.1 Activities

Shi and Deng (2000) developed a generic object-oriented data structure to represent an activity. This object-oriented expression of activity can represent the information of an activity at different levels and planning stages. For instance, it may indicate a work package at pre-construction stage. Therefore, activities can be further broken down into lower levels as more information are available when project is at construction stage. In this research, the object-oriented expression of activity is adopted to show the characteristics of a general resource-constrained project-scheduling problem as shown in Eq. (4.1).

$$\text{Activity } \{E, D, C, P, R, O, I, S\} \quad (4.1)$$

where,

- E*: Execution modes. Activities may have more than one execution mode. Each execution mode has its own set of resource requirements and estimated duration. For example, an excavation work may require 12 days if an average excavator is used or it may require 10 days if a more powerful excavator is used.
- D*: Activity duration.
- C*: Activity cost.
- P*: Precedence. Each activity has its precedence relationship. Usually an activity cannot start until all its predecessors have completed.
- R*: Resource demands. Resources required by each activity may be renewable or non-renewable. The costs of renewable resources are usually measured based on their hourly or daily rates. Labor and equipment are two examples of renewable resources. Non-renewable resources are mainly referred to raw materials.
- O*: Overlap. Although precedence relationships should be observed, overlapping is sometimes possible. Some activities can be defined to allow their successors to start,

Chapter 4. A Two-Phase GA Model for Resource-Constrained Project Scheduling

even when they are only partially completed. For example, the installation of temporary structure may begin when the excavation work is 80% completed.

I : Interruption. In a construction project, some activities have to be continuously executed once they are started, whereas some activities may be interrupted, i.e., their resources may be used by other activities for a while and then return to them. Interruption of an activity by the other activity is due to the relative importance between these two activities. In view of limited resources, preempting resources from non-essential or non-urgent ongoing activities for essential or urgent ones may shorten duration and/or smooth resource requirements. Thus, it is reasonable to assume that only ongoing activities are eligible for interruption. It is also assumed that when an interrupted activity is resumed, the working mode is kept the same for simplicity.

S : State of the activity.

A construction activity could be in one of the three states: scheduled but not started (SC), ongoing (ON) and completed (CO), i.e.,

$$S = \{SC, ON, CO\} \quad (4.2)$$

The precedence relationships within a construction project can be classified into four categories: FS (finish-to-start), SS (start-to-start), SF (start-to-finish) and FF (finish-to-finish). Thus,

$$T = \{FS, SS, SF, FF\} \quad (4.3)$$

For simplicity, only the FS (finish-to-start) relationship between activities is demonstrated in the research. Shortly in the future, all the other three types of relationships will also be incorporated into the computer system.

Chapter 4. A Two-Phase GA Model for Resource-Constrained Project Scheduling

Resources can be further broken down to renewable (RE) and nonrenewable (NR) resources. Renewable resources can be classified as labor (L) and equipment (E), as shown in Eq. (4.4).

$$R = \{RE, NR\}; RE = \{L, E\} \quad (4.4)$$

Furthermore, activity cost consists of resource cost (RC) and interruption cost (IC). An ongoing activity may be interrupted and restarted later, with some cost or increase in estimated time for completion. For example, the excavation work on site A can be interrupted with the excavators being moved to site B for more urgent excavation work. A cost will be incurred to reflect the cost of interrupting the current work, such as the cost for workers to familiarize the new work condition, the cost for moving the excavators to site B and so on. Correspondingly, a similar cost will be incurred to reflect the cost of resuming the interrupted work. While such cost is realistically tied to the interrupted activity, the use of interruption cost in a lump-sum value may be considered to simplify the problem. Hence the activity cost is expressed as follows:

$$C = \{RC, IC\} \quad (4.5)$$

Activity objects can be instantiated through object-oriented representations by supplying the information to the corresponding attributes. The information can be updated with project progress. Subsequently, the work schedule can be modified or updated according to the updated information. An example activity object is illustrated in Figure 4.1.

Chapter 4. A Two-Phase GA Model for Resource-Constrained Project Scheduling

Fig. 4.1 Example activity object

4.2.2 Resources

In practice, resources are limited in quantity. As allocation and arrangement of renewable resources are the concern of this paper, non-renewable resources (like raw materials, etc.), which could usually be estimated as a fixed amount based on the quantities needed, are not discussed in detail here. The resource constraint pattern for renewable resources can be defined mathematically as follows:

$$(Ra_t - Rr_t) \geq 0, 1 \leq t \leq T \tag{4.6}$$

Chapter 4. A Two-Phase GA Model for Resource-Constrained Project Scheduling

$$Rr_t = \sum_{j=1}^A Rr_{tj} \quad (4.7)$$

where t is the current time; T is the total duration; Ra_t is the resource available at time t ; Rr_t is the resource required at time t ; Rr_{tj} is the resource required at time t by activity j ; $j \in A$, the set of activities scheduled at time t .

4.3 A Two-Phase GA Model for Resource-Constrained Project Scheduling

In this research, the ideas from the aforementioned research (time-cost trade-off analyses and resource scheduling) were integrated to form a new approach using a two-phase genetic algorithm (GA) to tackle resource-constrained project scheduling problems (RCPS). In this model, a GA-based time-cost trade-off analysis is used to determine which execution mode should be selected for each activity. A set of constraints (e.g., precedence relationships, resource demands and availability, interruption and overlapping of activities, etc.) are provided as the criteria for determining the scheduling of each activity and a GA-based resource scheduling process will check the satisfaction of the constraints after the execution modes for all the activities are selected through the time-cost trade-off analysis.

4.3.1 Chromosome Representations for Resource-Constrained Project Scheduling Problems

Three issues are involved in choosing a suitable chromosome representation: the choice of decision variable being coded, the mapping from activity to gene position on the chromosome, and the form of coding to be used for gene values.

Chapter 4. A Two-Phase GA Model for Resource-Constrained Project Scheduling

4.3.1.1 Choice of Decision Variable

The choice of the decision variable tends to be problem dependent. An improper choice of the decision variable can cause a lot of ad hoc rework.

In resource scheduling, two alternatives suggest themselves: the genes could represent either the start times of the activities or the priority with which to schedule activities. If the first alternative is chosen, the precedence relation is often violated, thus requiring external repair. This is because, in this alternative, the precedence relationship between the activities is not encoded in the chromosome and the genetic algorithm has to learn through exploration with the start times, which is inefficient. In the other alternative, the genes values are real numbers which code for the scheduling priority of an activity. In this research, the second alternative was adopted for representing the resource scheduling.

4.3.1.2 Activity-to-Gene Mapping

In time-cost optimization, which is a kind of multiple-objective optimization, the genes usually represent the duration of the corresponding activities.

In the time-cost optimization, it would be straightforward to use a simple mapping of activity-to-gene position with the genes representing the activity duration; in such mapping, the first activity, i.e., the activity's ID is 1, would occupy the first gene position and so on.

However, in the resource scheduling, the need to consider the activity-to-gene mapping arises because that a chromosome is a linear string of genes, whereas a project network has some forms of structure, principally induced by the precedence relationships between the activities. The mapping based on a topological sorting of the activities performs better than the simple mapping described above.

Chapter 4. A Two-Phase GA Model for Resource-Constrained Project Scheduling

In this case, activities are first mapped using the simple mapping with the genes representing the priority of the activity; then activities are sorted using precedence relationships as the criterion. Activities of the same topological rank are then mapped to a contiguous segment of the chromosome. In the segment where the genes have the same topological rank, the genes are sorted from high priority to low priority based on the priority of the activities assigned. The precedence constraint is met through this kind of mapping.

The following gives an example of topological sorting:

An instance of a project can be represented using a directed acyclic graph (DAG). A directed acyclic graph $G = (V, A)$ consists of a set of nodes V representing activities ID and a set of directed edges A representing the precedence constraints among activities. Figure 4.2 gives an example of a DAG to represent a project. The problem of ranking all activities in an appropriate order to meet the precedence constraint then is equivalent to the problem of generating a topological sorting of the DAG as defined below.

For a given direct graph $G = (V, A)$, a topological sorting is a linear ordering of all its nodes such that for any directed edge $(u, v) \in A$, node u appears before node v in the ordering. Figure 4.3 shows a topological sorting of the example given in Figure 4.2. Nodes are arranged such that all directed edges go from left to right.

Fig. 4.2 Network representation of a project in directed acyclic graph (DAG)

Fig. 4.3 Topological sorting for Figure 4.2

4.3.1.3 Form of Coding

Choosing an appropriate representation of candidate solutions to the problem is the foundation for applying the genetic algorithms to solve the real world problems, which conditions all the subsequent steps of genetic algorithm. Two widely used forms of coding for gene values are the binary and real number representation. GA traditionalists prefer binary coding. With binary variables, the parameter range is discretized with enough resolution to enable the output to be affected to the desired level of precision. The discretized value is then represented as a bit string consisting of an appropriate number of 1 and 0 bits. For example, to represent a variable in the range of (a, b) , the required bits (denoted with m_j) for this variable is calculated as follows: $2^{m_j-1} < (b - a) \leq 2^{m_j} - 1$. However, coding becomes more difficult when the number of values is not a power of 2. For example, to represent a number between 0 and 10, it will required 4 bits to cover this range; but number 11 to 15 is outside the range, hence, it is problematic since the assignment of these extra values will bias the search in an unpredictable way. The use of real number representation has been a more recent development. Especially for the problems from industrial engineering world, the binary representation is difficult to apply directly because the binary string is not a natural coding. Hence the selection of form of coding is subject to the conditions of the problems.

4.3.2 Proposed Two-Phase GA Model for Resource-Constrained Project Scheduling

The operational architecture of the two-phase GA model for resource-constrained project scheduling problems is shown in Figure 4.4. The model consists of four subsystems: the input subsystem, the time-cost trade-off subsystem, the resource scheduling subsystem

Chapter 4. A Two-Phase GA Model for Resource-Constrained Project Scheduling

and the output subsystem. All the four subsystems are described in detail in the following sections.

4.3.2.1 Input Subsystem

To initialize the resource-constrained project scheduling, two types of information are required: GA-related input information and project input information. The GA-related input information includes population size, crossover rate, mutation rate, maximum generation, etc. The project-related input information includes estimates of direct costs, durations and resource demands for each corresponding execution mode, state of activity, the precedence relationships between activities and other constraints and the indirect cost rate for the project. The input subsystem provides interfaces for users to input or update information and stores the information data for further scheduling.

Chapter 4. A Two-Phase GA Model for Resource-Constrained Project Scheduling

Fig. 4.4 Operation of two-phase GAs model

Chapter 4. A Two-Phase GA Model for Resource-Constrained Project Scheduling

4.3.2.2 Time-Cost Trade-Off Subsystem

Feng et al. (1997) treated construction time-cost trade-off problems as multiobjective optimization problems, which try to minimize cost and duration simultaneously, and proposed a model using genetic algorithms and the Pareto front approach to solve the problems. In this research, a GA-based multiobjective optimization technique using the Pareto approach was adopted for the time-cost trade-off analysis. It should be mentioned that the costs referred here are the direct costs. The indirect costs are included in the output subsystem (see section 4.3.2.4).

This approach used the following techniques to obtain a time-cost trade-off curve:

- Using the concept of Pareto dominance to assign scalar fitness values to individuals;
- Using the concept of elitism to make those individuals that represent a nondominated front in that run of generation participate into next generation; and
- Using random selection concept to preserve the diversity in the population.

A pool of chromosomes is created to represent possible execution modes. Each gene in a chromosome represents the execution mode of its corresponding activity. Once the execution modes for each activity are decided, the corresponding activity cost, duration and resource demands will be determined. Afterwards, the execution modes will be input to the resource scheduling subsystem, which will check the satisfactions of the constraints and produce a feasible schedule. As a result, the total project duration and cost for each execution-mode chromosome are fed back to the time-cost trade-off subsystem for evaluation. The subsystem uses one-point crossover and random mutation operators to generate feasible child chromosomes. According to the objective functions described later in details, the fitness for each chromosome is calculated. The surviving chromosomes for the next generation are selected according to the roulette wheel principle. This means that the selection possibility for a chromosome i is proportional to the ratio of $\frac{1}{f_i} / \sum_{j=1}^{pop_size} \frac{1}{f_j}$,

where f_i is the fitness value of chromosome i . Note that fitness is to be minimized here,

Chapter 4. A Two-Phase GA Model for Resource-Constrained Project Scheduling

i.e., small fitness values correspond to high selection probabilities. The elitist selection method is combined with the selection procedure to preserve the best chromosomes for the next generation, thus all the nondominated chromosomes are automatically selected into the next population to overcome the stochastic error of sampling. In the final step of time-cost trade-off subsystem, the optimal or sub-optimal solution is exported to the output subsystem.

The flow of the algorithm in time-cost trade-off subsystem is as follows:

Symbol:

P_t (Population)

N (Population size of the time-cost trade-off subsystem)

T_{tc} (Maximum number of generation of the time-cost trade-off subsystem)

C_{tc} (Crossover rate of the time-cost trade-off subsystem)

M_{tc} (Mutation rate of the time-cost trade-off subsystem)

P', P'', P''' (Mating Pool)

$F(i)$ (fitness value assigned to individual i)

Terminology:

- Trade-off curve: The nondominated set of a generation.
- Chromosome structure: each solution is encoded by a binary string, the length of which corresponds to the number of choices of different duration available. Each choice is assigned a position within the binary string, where 0 means the choice is not in the solution, 1 stands for “the solution contains the corresponding choice”. (See Figure 4.5 for configuration). Note that the execution modes for ongoing activities do not participate in mutation, i.e., the execution mode for an ongoing activity can not be changed.

Chapter 4. A Two-Phase GA Model for Resource-Constrained Project Scheduling

Position: activity ID

Value: 1 (the corresponding choice is selected) or 0 (the corresponding choice is not selected) (Note: for each activity, one and only one choice of durations can be selected; please refer to Table 4.1 for information of activities)

1			2		3			4		5		6		7		8		Position	
1	0	0	1	0	0	0	1	0	1	0	1	0	1	0	0	1	0	1	Value

Fig. 4.5 Binary encoding

Fig. 4.6 Crossover operator used in time-cost trade-off subsystem

Fig. 4.7 Mutation operator used in time-cost trade-off subsystem

Procedure

- Step 1: **Initialization:** Set $t = 0$; Randomly select N individual binary strings. Each individual string represents a certain network schedule without resource constraints. These N strings form the parent generation.
- Step 2: **Calculation of resource-constrained project duration and cost:** For $i = 1, \dots, N$, do
- Since chromosome i has set which mode for each activity is selected for execution, the activity information for respective execution mode, including estimated activity duration, resource requirements, cost, interruption and/or

Chapter 4. A Two-Phase GA Model for Resource-Constrained Project Scheduling

overlap settings, are inputted as sets of data to resource scheduling subsystem.

- b) Execute the genetic algorithms in resource scheduling subsystem. The genetic algorithms will be described later. An optimal schedule which satisfies all project constraints can be generated as the result of genetic evolution in the resource scheduling subsystem.
- c) The project total duration and cost calculated from the optimal and feasible schedule are used for evaluation of the fitness of chromosome i

Step 3: ***Fitness assignment:*** First calculate the ranking of individuals in P_t by using Goldberg's suggestion (Section 3.3.3.1.2). Thus, a non-dominated sorting is used to rank a search population according to Pareto optimality. First, non-dominated individuals in the population are identified. They form the trade-off curve of that run of generation. They are given the rank 0 and are removed from the population. Then the non-dominated individuals in the reduced population are identified, given the rank 1, and then they are also removed from the population. This procedure of identifying non-dominated sets of individuals is repeated until the whole population has been ranked, as depicted in Figure 4.8.

For those nondominated chromosomes, the fitness is calculated as $f(i) = \frac{m}{N+1}$,

where m is the number of chromosomes that chromosome i dominates, N is the population size. For other chromosomes, the fitness is calculated as $f(i) = 1 + n$, where n is the sum of all the fitness values of the nondominated chromosomes that dominate chromosome i . For example, as depicted in Figure 4.8, the non-dominated chromosome A dominates chromosomes B and C, thus the fitness of chromosome A is calculated as $\frac{2}{13+1} = 0.143$, while the fitness of chromosome

C, which is dominated by chromosome A only, is calculated as $1 + 0.143 = 1.143$. (Note that fitness is to be minimized here, i.e., small fitness values correspond to high reproduction probabilities):

Chapter 4. A Two-Phase GA Model for Resource-Constrained Project Scheduling

To determine the rank between the individuals among the population, the following steps were used:

- Given any two individuals A, B of the population. A is said to dominate B iff the cost of A is less than or equal to the cost of B and the duration of A is less than or equal to the duration of B; and A is better in one objective, i.e., either the cost is less or the duration is less. Mathematically, the rule can be expressed as following:

$a \succ b$ (a dominates b) iff

$$a) \text{Cost}_A \leq \text{Cost}_B \wedge \text{Duration}_A < \text{Duration}_B$$

or

$$b) \text{Cost}_A < \text{Cost}_B \wedge \text{Duration}_A \leq \text{Duration}_B$$

- An individual A is denoted as *nondominated* regarding a given population iff no individual of the population dominates A.
- Those solutions that are nondominated over the entire search space form the trade-off curve of that generation.

Fig. 4.8 Trade-off curve and population ranking based upon non-dominated sorting

Step 4: **Selection:** Set $P' = 0$. Select and put all the nondominated individuals into P' .

Then, remove them from P_t . Assume the number of nondominated individuals is N_{T-O} . Thus, for $i = 1, \dots, N - N_{T-O}$, do

- a) Select one individual $i \in P_t$ at random

Chapter 4. A Two-Phase GA Model for Resource-Constrained Project Scheduling

- b) To keep the diversity of the population, a random selection probability is predetermined. If the random number which is between 0 and 1 generated by computer is greater than the random selection probability, then $P' = P' + \{i\}$; else
- c) If $F(i)$ is less than a random number generated by computer which is between smallest and the largest fitness of that population, then $P' = P' + \{i\}$; else go back to step (a). Note that fitness is to be minimized here and the smaller a chromosome's fitness the greater probability that it will be selected to reproduce.

Step 5: **Recombination**: Set $P'' = 0$. For $i = 1, \dots, \frac{N}{2}$, do

- a) Choose two individuals $i, j \in P'$ and remove them from P' .
- b) Recombine i and j using the crossover operator illustrated in Figure 4.6. The resulting children are k, l .
- c) Add k, l to P'' if the random number generated by the computer is greater than the crossover rate C_{ic} . Otherwise, add i, j to P'' .

Step 6: **Mutation**: Set $P''' = 0$. For each individual $i \in P''$, do

- a) Mutate i with mutation rate M_{ic} using the mutation operator illustrated in Figure 4.7. The resulting individual is j . Note that the chromosomes with fitness less than 1, i.e., non-dominated individuals, are not mutated, while the chromosomes with largest fitness are always mutated.
- b) Set $P''' = P''' + \{j\}$.

Step 7: **Termination**: Set $P_{t+1} = P'''$ and $t = t + 1$. If $t \geq T_{ic}$ or the trade-off curve remain the same after a pre-specified number of iterations, stop the algorithm. Otherwise, go to Step 2.

Chapter 4. A Two-Phase GA Model for Resource-Constrained Project Scheduling

4.3.2.3 Resource Scheduling Subsystem

Davis (1991) advocates hybridizing the GA with the existing problem-solving algorithms so that the domain expertise will be preserved. Many successful applications have been reported in the domain of resource scheduling (Chan et al. 1996; Gen and Cheng 1997). The GA performs the basic GA processes of selection, recombination and mutation on succeeding populations of solutions, while the evaluation function for the resource allocation problem is supplemented with conventional heuristic methods

In the field of resource scheduling, most of the heuristic methods known so far can be viewed as priority dispatching rules that assign activity priorities in making sequencing decisions for resolution of resource conflicts according to either temporal or resource-related heuristic rules. Gen and Cheng (1997) incorporated priority dispatching ideas in a new approach using genetic algorithms to overcome some of the previously mentioned difficulties associated with mathematical and heuristic methods.

In Gen and Cheng's (1997) approach, the topological-based heuristics, i.e., the heuristics based on the precedence between activities in the schedule, are used. In this approach, priority for each activity is firstly generated by GA. The activities are then sorted using precedence relationships as the criterion. Activities of the same topological rank are then mapped to a contiguous segment of the chromosome. In the segment where genes have the same topological rank, the genes are sorted from high priority to low priority based on the priorities assigned to the activities. Thus, the precedence constraint is met. Because a topological sort gives a feasible order of activities, a schedule can be constructed by selecting activities in the order of their appearance in the topological sort and scheduling them one at a time as early as resource availability allows.

In this research, the Gen and Cheng's (1997) approach was employed for resource allocation, but with some modifications as follows:

Chapter 4. A Two-Phase GA Model for Resource-Constrained Project Scheduling

- Activities which have started are updated with remaining duration, cost, and nonrenewable resource requirements (raw materials). It is reasonable to assume that renewable resources (labor and equipment) demands remain unchanged.
- Activities which have started and required continuity must be assigned with highest priorities so that resources can be allocated to such activities first. For example, if there are m ongoing activities that can not be interrupted among total n activities, then priority from n to $n-m+1$ will be randomly assigned to those activities, where n represents the top priority. Activities which have started and allow interruption will be treated as non-started activities.
- Activities whose predecessors allow overlapping can start earlier according to the overlap time allowed if not violate resource constraints.

The flow of the procedure is shown in Figure 4.9. In the resource scheduling subsystem, each gene in a chromosome represents the priority of its corresponding activity. The subsystem uses one-point crossover and uniform mutation operators. The scheduled project duration is taken as the fitness value of each chromosome and the roulette wheel approach is adopted as the selection procedure. The elitist selection is incorporated as well.

The resource scheduling subsystem is further described in the following sections in three aspects: the concept of priority-based encoding, procedure for scheduling the resource-constrained activities, and the flows of genetic algorithms in the resource scheduling subsystem.

Chapter 4. A Two-Phase GA Model for Resource-Constrained Project Scheduling

Chapter 4. A Two-Phase GA Model for Resource-Constrained Project Scheduling

Fig. 4.9 The flow of procedure in resource scheduling subsystem

4.3.2.3.1 Priority-Based Encoding

Recall that a gene contains two kinds of information: the position of a gene located within the structure of a chromosome, and the value taken by the gene. Thus there are two possible ways to represent an activity: using the position and the value. Here the position is used to denote the activity ID, and the value is used to denote the priority associated with the activity, as shown in Figure 4.10. The value of a gene is an integer exclusively within $[1, n]$, where n is the number of activities. The larger the integer is, the higher the priority is. Thus, activity 1 has n number of possible priorities, and activity 2 has $n - 1$ number of possible priorities, and activity 3 has $n - 2$ number of possible priorities, and so forth. Then the total number of possible combinations is equal to $n!$. For a project with numerous activities, the number of possible combination will be a giant number. The genetic algorithms can be used here to search only a small portion of the possible solution space. Despite the low percentages of search space versus the total solution space, the accuracy can be expected to be impressively high. As an advantage of genetic algorithms, it is a good balance of computational effort and accuracy.

Position: activity ID
Value: priority of activity (Note: Largest value stands for highest priority)

1	2	3	4	5	6	7	8	Position
3	4	6	7	8	1	2	5	Value

Fig. 4.10 Priority-based encoding

A one-pass procedure is used to generate a topological sorting from a chromosome: to determine an activity from left to right in a single pass. When making a decision regarding a position, several activities may compete for the position, and the one with the highest priority wins the position. The encoding does not explicitly represent a

Chapter 4. A Two-Phase GA Model for Resource-Constrained Project Scheduling

topological sorting for a given Directed Acyclic Graph (Section 4.3.1.2). It just contains some messages for the resolution of conflicts. A topological sorting can be determined uniquely according to the encoding in most cases. Any changes in priorities usually result in a different topological sorting. Therefore, this encoding is essentially capable of representing all possible topological sorting of a given Directed Acyclic Graph.

The procedure to generate a topological sorting from the encoding is as follows. Consider an example shown in Figure 4.11 with the priority of the activities given in Figure 4.12. An array $A[.]$ is used to store the topological sorting generated. At the beginning $A[1] = 1$. Then activity 1 is removed from the diagram and the arcs pointing from activity 1 to other activities are removed as well, as shown in Figure 4.13. Two activities 2 and 3 compete for $A[2]$. Their priorities are 4 and 6 respectively. Activity 3 wins the position because it has the higher priority. After fixing $A[2] = 3$, the candidates for next position $A[3]$ are activity 2 and 6 after removing activity 3 and the arcs connecting from activity 3 to other activities. As activity 5 has an arc pointing to it from activity 4, it can not be an eligible candidate. Activity 2 wins the position and fix $A[3] = 2$. Repeat these two steps:

- Step 1: Construct the set of candidates for current position. The criterion to be an eligible candidate is that the activity has no preceding activities at that moment, i.e., not arcs pointing to it.
- Step 2: Select the highest-priority activity and remove the activity and all the arcs connecting from it to other activities. Repeat this step until a topological sorting is obtained, as shown in Figure 4.14.

Chapter 4. A Two-Phase GA Model for Resource-Constrained Project Scheduling

Fig. 4.11 Network representation of a project in Directed Acyclic Graph (DAG) (Same as Fig 4.2)

1	2	3	4	5	6	7	8	Position
3	4	6	7	8	1	2	5	Value

Fig. 4.12 Activity priority in accordance with activities in Fig 4.11

Fig. 4.13 Network representation of a project after activity 1 is removed

Fig. 4.14 Topological sorting of the Directed Acyclic Graph (DAG) in Fig. 4.11

Chapter 4. A Two-Phase GA Model for Resource-Constrained Project Scheduling

4.3.2.3.2 Procedures for Scheduling Resource-Constrained Activities

During each generation, chromosomes are evaluated using some measure of fitness. The following three major steps are included in the evaluation phase:

- Step 1: Convert chromosomes to topological sortings.
- Step 2: Obtain the total project duration through generating a feasible schedule that satisfies all project constraints from the topological sortings.
- Step 3: Calculate fitness values for each schedule.

The procedure for Step 1 is described in Section 4.3.2.3.1. Because a topological sorting gives a feasible order of activities, a schedule can be constructed by selecting activities in the order of their appearance in the topological sorting and scheduling them one at a time as early as resource availabilities allow. In this report, only one resource is used for the testing purpose. In the proposed future work, a single-resource-constrained scheduling problem will be extended to a multiple-resource-constrained scheduling problem.

Let i be the iteration index of the procedure, P_j be the set of all direct predecessors of activity j , j be the project ID, $PS[.]$ be the array storing topological sorting, σ_j and θ_j be the start and finish times associated with activity j , d_j be the duration associated with activity j that is equal to $\theta_j - \sigma_j$, $b[l]$ be the array for storing the amount of resource available in day l , and r_j be the daily resource consumption associated with activity j . The procedure for determining finish time for each activity from a given topological sorting is given below.

Procedure

Step 1: **Initialization.**

$i = 1$ (Iteration index)

$j \leftarrow PS[1]$ (The first activity in the topological sorting will be the starting activity)

Chapter 4. A Two-Phase GA Model for Resource-Constrained Project Scheduling

$\sigma_j = 1, \theta_j = d_j$ (Start and finish days for the first activity j)

$b[l] = b[l] - r_j, l = 1, 2, \dots, d_j$ (Resources consumed by the first activity j are deducted from the resource array)

Step 2: *Termination Test.*

If $i = m$ (m is the number of activities in topological sorting), go to Step 5;
else $i = i + 1$ and go to Step 3.

Step 3: *Determination of the finish time of activity j*

$j \leftarrow PS[i]$

$\sigma^{\min} = \max \{ \theta_k \mid k \in P_j \}$ (The start day of activity j should not be earlier than the latest finish day of its predecessors, where k is the activity in P_j and θ_k is the finish time of activity k).

Let $t = \sigma^{\min} + 1$ and start a loop with the iteration index n initializing from 0 and an iteration number of d_j :

For $(n = 0, n < d_j, n = n + 1)$, do

a) If $b[t + n] < r_j$, then, $t = t + n + 1$ and restarting the loop with $n = 0$

b) If $b[t + n] \geq r_j$, then continue the loop

(The amount of resource used by activity j should not exceed the available quantity at any time. The flow of this algorithm is plotted in Figure 4.15)

$\sigma_j = t$ and $\theta_j = \sigma_j + d_j - 1$

Step 4: *Update of the available resource.*

$b[l] = b[l] - r_j, l = t, \dots, t + d_j - 1$ and go back to Step 2

Step 5: *Stop.*

Return the total project duration, i.e., $\theta_{j(i=n)}$, where activity j is the last activity in the topological sorting.

Chapter 4. A Two-Phase GA Model for Resource-Constrained Project Scheduling

An example is given to elaborate the procedure described above. Considering a project with network representation in Figure 4.11 and assigned activity priority in Figure 4.12, the array $PS[.]$ stores the topological sorting illustrated in Figure 4.14, i.e., $PS[1] = \text{Activity 1}, PS[2] = \text{Activity 3}, \dots, PS[8] = \text{Activity 8}$. Assuming that the resource considered here is the equipment and there are totally 7 equipments allocated in site, hence each element in array b is 7, i.e., $b[1] = b[2] = \dots = b[m] = 7$, where m is the estimated project duration. The resource consumption rate and the duration of each activity are tabulated in Table 4.1.

The first element in $PS[.]$ is activity 1, thus in the initialization step, $\sigma_1 = 1$, $\theta_1 = 2$, and the remaining available resource for day 1 and day 2 is $b[1] = b[2] = 7 - 5 = 2$. Then i is increased to 2, the second element in $PS[.]$, i.e., $PS[i = 2]$ is activity 3. Thus, $\sigma_3 = 2 + 1 = 3$, $\theta_3 = 3 + 2 - 1 = 4$, and the remaining available resource for day 3 and day 4 is 2. When i is increased to 3, as the third element in $PS[.]$, activity 2 can not start until day 5, as there is a resource shortage of 2 on day 3 and day 4. The scheduling of remaining activities is similar. Finally, the schedule of project and the resource profile are plotted in Figure 4.16 and Figure 4.17 respectively.

Table 4.1 Duration and resource consumption rate of each activity in the example

Activity ID	1	2	3	4	5	6	7	8
Duration (day)	2	3	2	3	1	3	1	1
Resource Consumption Rate (unit/day)	5	4	5	3	2	3	6	6

Chapter 4. A Two-Phase GA Model for Resource-Constrained Project Scheduling

Fig. 4.15 The flow chart for Step 3

Fig. 4.16 The schedule of the example project

Chapter 4. A Two-Phase GA Model for Resource-Constrained Project Scheduling

Fig. 4.17 Resource profile of the example project

The total project duration is the objective value. Since a minimization is being dealt with here, a fitness value will be assigned to each chromosome in accordance with the objective value in order to ensure that the fitter chromosome has a better opportunity to be selected in to next generation. Here, the fitness value is equal to the total project duration. The smaller the fitness value is, the fitter the individual is.

4.3.2.3.3 Flow of the Genetic Algorithms in Resource Scheduling Subsystem

In general, the flow of the genetic algorithms in resource scheduling subsystem is as follows.

Symbol:

***D[.]** (The array storing selected duration for each activity obtained from time-cost trade-off subsystem)*

***R[.]** (The array storing resource consumption for each activity obtained from time-cost trade-off subsystem)*

***O[.]** (The array storing overlap day for each activity obtained from time-cost trade-off subsystem)*

***S[.]** (The array storing state for each activity obtained from time-cost trade-off subsystem. An activity is in either of three states: completed, ongoing and not started)*

Chapter 4. A Two-Phase GA Model for Resource-Constrained Project Scheduling

$I[i,j]$ (The array storing interruption preference (yes or no) for each activity obtained from time-cost trade-off subsystem.)

$C_r[i,j]$ (The array storing resource cost for each activity obtained from time-cost trade-off subsystem)

$C_i[i,j]$ (The array storing interruption cost for each activity obtained from time-cost trade-off subsystem)

Note that ongoing activities are updated with remaining duration and resource cost. It is reasonable to assume that renewable resources (labor) demands remain unchanged.

N_{rs} (Population size in resource scheduling subsystem)

T_{rs} (Maximum number of generation in resource scheduling subsystem)

C_{rs} (crossover probability in resource scheduling subsystem)

M_{rs} (mutation rate in resource scheduling subsystem)

Fig. 4.18 Crossover operator used in resource scheduling subsystem

Chapter 4. A Two-Phase GA Model for Resource-Constrained Project Scheduling

Fig. 4.19 Mutation operator used in resource scheduling subsystem

Procedure

Step 1: **Initialization:** Set $t = 0$; Randomly select N_{rs} individuals to form the parent generation. Each individual is in the form as described in Section 4.3.2.3.1. To generate an individual chromosome, the following steps are used:

- Read data from $S[.]$ and $I[.]$. Set the length of the chromosome to $A_p - A_f$, where A_p is the number of activities in the project, A_f is the number of the completed activities.
- Randomly assign priorities from $A_p - A_f$ to $A_p - A_f - m + 1$ to ongoing and non-interruptible activities, where m is the number of ongoing activities that can not be interrupted, $A_p - A_f$ represents the top priority.
- Randomly assign priorities from $A_p - A_f - m$ to 1 for the left activities, which are non-started activities and ongoing activities that allow interruption.

Step 2: **Fitness Assignment:** Calculate the fitness value of each individual in the population by invoking the procedures described in Section 4.3.2.3.2.

Step 3: **Selection:** ... Same as the selection method described in Section 4.3.2.2. Note that the chromosomes with shortest durations are automatically selected into next generation.

Step 4: **Recombination:** ... Same as the recombination method described in Section 4.3.2.2 but adopting the crossover operator illustrated in Figure 4.18

Step 5: **Mutation:** ... Same as the mutation method described in Section 4.3.2.2 but adopting the mutation operator illustrated in Figure 4.19. Note that the

Chapter 4. A Two-Phase GA Model for Resource-Constrained Project Scheduling

chromosomes with shortest durations are not mutated, while the chromosomes with longest durations are always mutated.

Step 6: **Termination:** Set $P_{t+1} = P'''$ and $t = t + 1$. If $t < T_{rs}$ then go to Step 2, else choose the chromosome that has shortest duration as the output chromosome. Thus, the total project cost by summing up all activities' resource cost and interruption cost if interrupted. The project schedule can be generated as well.

4.3.2.4 Output Subsystem

In the output subsystem, the direct costs and their corresponding durations, which are generated by the time-cost trade-off and resource scheduling subsystems, are gathered for further plotting. For instance, the final generation is shown in Figure 4.20. Subsequently, the trade-off curve between direct cost and duration is extracted from the final generation (see Figure 4.21). After finding the trade-off curve, construction planners can determine the total cost by summing up the estimated indirect cost and the direct cost from the trade-off curve. Indirect cost is usually assumed to be proportional to the project duration. As shown in the Figure 4.22, the optimal choice to perform the project would be the lowest total cost. Using trade-off curve as the objective function allows for much more efficient evaluations of various indirect cost rates without performing another GA run. This is an improvement over treating the total cost as the objective in the GA. What's more, the feasible schedule and resource profile for corresponding combination of project duration and cost can be obtained as well.

Chapter 4. A Two-Phase GA Model for Resource-Constrained Project Scheduling

Fig. 4.20 The final generation

Fig. 4.21 The optimal tradeoff curve

Fig. 4.22 The optimal choice

4.4 Example Project

Many test cases were generated to verify the accuracy of the algorithm. In this section, a simple project is planned with the network shown in Figure 4.23 and activity data in Table 4.2. The GAs parameters, such as crossover rate and mutation rate, are tuned through experimental tests to obtain good performance using GAs search, as tabulated in Table 4.3. As only one type of resource is used in this example, the cost of each activity is equal to the product of the amount of resource unit consumed and the unit pay rate which is assumed as \$100 per unit. It is pointed that the reduction of the duration of an activity, when feasible, usually comes at a price, since it typically involves an increase in the resource utilized, for example, using more equipments. Thus, shortening the duration will usually result in an increase in the amount of resource unit consumed as well as the cost. The indirect cost is set at \$2,200 per day.

Fig. 4.23 The network of example project

Chapter 4. A Two-Phase GA Model for Resource-Constrained Project Scheduling

Table 4.2 Activity information

Activity ID	Execution mode	Duration (day)	Labor requirement ^a (man)	Interruption	Direct cost ^b		Overlap ^d (day)
					Resource cost ^c (\$)	Interruption cost (\$)	
1	1	5	15	Yes	7,500	100	0
2	1	4	16	No	6,400	0	0
	2	6	10	No	6,000	0	0
	3	8	7	No	5,600	0	0
	4	9	6	No	5,400	0	0
3	1	6	13	Yes	7,800	100	0
	2	8	9	Yes	7,200	100	0
	3	10	7	Yes	7,000	100	0
4	1	12	16	No	19,200	0	1
	2	15	10	No	15,000	0	1
	3	18	8	No	14,400	0	1
5	1	22	18	No	39,600	0	0
	2	24	16	No	38,400	0	0
	3	26	14	No	36,400	0	0
	4	28	12	No	33,600	0	0
6	1	14	20	No	28,000	0	0
	2	18	15	No	27,000	0	0
	3	24	8	No	19,200	0	0
7	1	9	17	No	15,300	0	0
	2	10	14	No	14,000	0	0
8	1	14	7	No	9,800	0	1
	2	15	6	No	9,000	0	1
	3	16	4	No	6,400	0	1
9	1	15	5	No	7,500	0	1
	2	18	4	No	7,200	0	1
	3	20	3	No	6,000	0	1
10	1	3	4	No	1,200	0	0
	2	5	2	No	1,000	0	0

a Maximum labor capacity: $L \leq 30$ men

b Indirect cost = \$2,200 per day

c Resource cost = Duration* Labor requirement*\$100/man/day

d Overlap days are the days allowed to be overlapped by its successors

Table 4.3 GAs parameters value

	Population	Generation	Crossover rate	Mutation rate
time-cost trade-off subsystem	300	16	0.4	0.02
resource scheduling subsystem	100	20	0.3	0.3

To illustrate the influences of constraints (e.g., resources constraints, interruption or/and overlap of activities) over the time-cost optimization, final generation of the GA calculation, optimal time-cost trade-off curve, total cost curve and the schedule for the optimal choice were plotted for the three types of conditions described below respectively (see Figures 4.24-35).

Chapter 4. A Two-Phase GA Model for Resource-Constrained Project Scheduling

4.4.1 Planning Project with Precedence Relationships Only

If only precedence relationships are active, the project duration and cost generated by resource scheduling subsystem is comparable to the one obtained from CPM computation. The optimal project total cost is \$227,700 with project duration of 49 days (see Figures 4.24-27).

4.4.2 Adding Resource Capacities into Active Constraints

If both precedence relationships and resource capacities are selected as the active constraints, the resource-constrained time-cost trade-off and the total cost curve were plotted in Figure 4.29 and 4.30 respectively. Due to the limitation of the resource availability, it can be seen that optimal project total cost is increased to \$244,000 with optimum project duration postponed to 56 days (see Figures 4.28-31).

4.4.3 Allowing Interruption for Ongoing Activities and Overlap of Selected Activities

For the example project described above, we assume that only activities 1 and 3 are started when the project has been commenced for one day. Activities 4, 8 and 9 allow overlap by their successors. The resulting resource-constrained time-cost trade-off and total cost curve were plotted in Figure 4.33 and 4.34 respectively. As the result of the interruption and/or overlap of activities, it can be seen that optimal project total cost is reduced to \$237,300 though the optimum project duration remains at 56 days. In Figure 4.35, it can be seen that the ongoing activity 1 was interrupted to make more critical activity 2 be executed earlier and activity 10 overlapped its predecessor (activity 9) by 1 day.

Chapter 4. A Two-Phase GA Model for Resource-Constrained Project Scheduling

Fig. 4.24 Final generation of the example project with precedence relationships only

Fig. 4.25 Optimal trade-off curve of the example project with precedence relationships only

Chapter 4. A Two-Phase GA Model for Resource-Constrained Project Scheduling

Fig. 4.26 Optimal choice of the example project with precedence relationships only

Fig. 4.27 Schedule for duration of 49 days if planning project with precedence relationships only (hatched blocks stand for critical path)

Fig. 4.28 Final generation of the example project under resource constraint

Chapter 4. A Two-Phase GA Model for Resource-Constrained Project Scheduling

Fig. 4.29 Optimal trade-off curve of the example project under resource constraint

Fig. 4.30 Optimal choice of the example project under resource constraint

Fig. 4.31 Schedule for duration of 56 days if adding resource capacities into active constraints (hatched blocks stand for critical path)

Chapter 4. A Two-Phase GA Model for Resource-Constrained Project Scheduling

Fig. 4.32 Final generation of the example project with all the constraints activated

Fig. 4.33 Optimal trade-off curve of the example project with all the constraints activated

Fig. 4.34 Optimal choice of the example project with all the constraints activated

Chapter 4. A Two-Phase GA Model for Resource-Constrained Project Scheduling

Fig. 4.35 Schedule for duration of 56 days if activating all the constraints (hatched blocks stand for critical path and solid blocks stand for completed part of the activities)

It is fair to point out that GAs share one drawback with heuristic methods: it is not possible to know if an optimal result has been obtained. However, the results can be improved by increasing the size of the population and the iterations of the generations. The new GAs also showed its efficiency in solving resource-constrained project scheduling problem. In this example, there are 15552 ($1 \times 4 \times 3 \times 3 \times 4 \times 3 \times 2 \times 3 \times 3 \times 2$) possible schedules for the time-cost optimizations without any resource constraints, while there are 5.64×10^{10} ($15552 \times 10!$) possible schedules for time-cost optimization with resource constraints. The new GAs searched 4800 (300×16) and 9600000 ($300 \times 16 \times 100 \times 20$) possible different schedules, only a small portion of the solution space. It is a good balance of computational effort and accuracy.

4.5 Summary

This chapter has presented the characteristics of resource-constrained project scheduling problems and proposed a two-phase GAs model for resource-constrained project scheduling, which incorporated GA-based computational techniques for time-cost trade-off and resource scheduling. A set of constraints are provided in the model for project scheduling problems. If only precedence relationships are selected, the time-cost trade-off result is comparable to the work done by Feng et al. (1997). On the other hand, if more project constraints are applied, a corresponding practical time-cost trade-off result can be generated using this model. Allocations of limited resources, interruption and/or overlap of activities are allowed in the scheduling process.

Chapter 4. A Two-Phase GA Model for Resource-Constrained Project Scheduling

It is fair to point out that GAs share one drawback with heuristic methods: it is not possible to know if an optimal result has been obtained. Some existing algorithms (e.g., mathematical methods) can solve some forms of scheduling problems, but fail to do so when the activity number increases or more constraints are added. Hence, using GAs to solve project scheduling problems is a good way to find optimal or near-optimal solutions.

An example is shown in this chapter to demonstrate the two-phase GA model. In next chapter, an object oriented computer system that implements the two-phase GA model proposed in this chapter will be described.

CHAPTER 5 JAVA APPLICATION DESIGN FOR RESOURCE-CONSTRAINED PROJECT SCHEDULING

5.1 Introduction

The application was analyzed and described using an analysis model with use cases and a domain analysis. Subsequently, it was expanded to a design model that describes representative slices of a technical solution. Finally, it was coded in Java. The notation used for graphical representation of the object-oriented model is based on the new standard Unified Modeling Language (Fowler and Scott 1997).

5.2 Requirements

Typically, a representative of end users of the system writes the text requirement specification. For the project scheduling application, it looks like:

- It is a support system for project scheduling.
- The project engineer is an employee of a construction company who is responsible for project scheduling and whose work is supported by the system.
- The project engineer creates tasks and defines relationships among tasks to form a project.
- The project engineer creates resources, such as crew and materials, which are available for the project.
- The project engineer can easily update and delete information about the tasks and resources in the system.

Chapter 5. Java Application Design for Resource-Constrained Project Scheduling

- The project engineer applies genetic algorithms built in the system to generate a time-cost trade-off curve under project constraints, including resource constraints and interruption and/or overlap of tasks.
- The system can generate project schedules using Gantt charts based on the result of the resource-constrained time-cost trade-off analysis.
- The system can run on all popular technical environments, including UNIX, Windows, and OS/2, and has a modern graphical user interface (GUI).
- The system is easy to extend with new functionality.

5.3 Analysis

The analysis is intended to capture and describe all the requirements of the system, and to make a model that defines the key domain classes in the system (*What* is handled in the system).

In object-oriented modeling, large and complex problems are decomposed and modeled as a set of objects. An object often represents a real-world object such as physical object, a concept, or an abstraction with crisp boundaries and meaning for the problem at hand (Rumbaugh et al 1991). An Object encapsulates both data and functions. For example, an object representing a construction activity, in a scheduling system, should encapsulate the required scheduling data (e.g., start date and duration) and scheduling functions (e.g. the scheduling procedures). The set of objects, in an object-oriented model, should provide a precise abstraction of what is required to be done and how it can be done. To provide such an abstraction for resource-constrained project scheduling and identify its main requirements, requirement analysis and domain analysis are conducted.

5.3.1 Requirement Analysis

The first step in analysis is to figure out what the system will be used for and who will be using it. These are the use cases and actors, respectively. The use cases describe what the project scheduling system provides in terms of functionality: the functional requirement

Chapter 5. Java Application Design for Resource-Constrained Project Scheduling

of the system. A use-case analysis involves reading and analyzing the specifications, as well as discussing the system with potential users of the system.

The project engineers are the actors identified who are the users of the system. The project engineers use the system to obtain optimal project schedules which satisfy all the project constraints.

The use cases in the project scheduling system are:

- Create a project
- Update or remove project
- Add Task
- Update or remove task
- Add resource
- Update or remove resource
- Run the two-phase GA model (as described in Chapter 4)
- Generate time-cost trade-off (TCTO) curve
- Plot Gantt chart for the corresponding project duration from TCTO curve

The project scheduling system analysis is documented in a UML use-case diagram as shown in Figure 5.1.

Chapter 5. Java Application Design for Resource-Constrained Project Scheduling

Fig. 5.1 Actor and use cases

5.3.2 Domain Analysis

After completing the specification and the use cases and considering which concepts should be handled by the project scheduling system, analysis moves to domain analysis to itemize the domain (the classes in the system). A group of objects with similar properties (i.e., data), common behavior (i.e., functions), and common relationships to other objects can be grouped into a class. For example, all objects representing project activities such as excavation, foundation, and concrete can be grouped into one class named “Task”. Classes are usually documented in a class diagram along with their relationships. In UML, class diagrams are used to describe the context of the problem in terms of the physical and conceptual elements needed to describe and obtain a solution to a problem. Within a specific model, there are different groups of class diagrams. These groups are called packages.

This research focused on developing two packages to include domain classes according to their functionalities. The first group describes the plan-related classes needed to define

Chapter 5. Java Application Design for Resource-Constrained Project Scheduling

a complete project plan. The plan refers to the estimates of time and resource for each task, as well as the precedence relationships between tasks and other constraints, such as resource constraints and interruption and/or overlap of tasks. This package is called the “Plan” package. The second group describes those classes needed to model the Genetic Algorithm portion of the program. This is called the “GA” package. The “GA” package is for making optimal and feasible project schedules. The schedule consists of a set of assignments of resources to tasks at specific times and arranges the start times of tasks to meet all project constraints. The domain classes are defined with stereotype <<Business Object>>, which is a user-defined stereotype specifying that objects of the class are part of the key domain and should be stored persistently in the system.

5.3.2.1 Plan Package

The domain classes in the plan package are as follows: *Project*, *Task*, and *Crew*. They are documented in a class diagram along with their relationships, as shown in Figure 5.2. The focal point in construction engineering and management is the construction project. A project is defined for every collection of related tasks and crews that needed to be completed under certain constraints. The “*Project*” class is to model the general characteristic of a construction project (e.g., project name and project organization) and perform needed functions at the project level such as initiating the functions performed by other associated classes (e.g., initiating the schedule optimization), saving and opening project data. A “*Task*” class encapsulates the necessary data (e.g., task name, duration, status, resource requirements) and functions (e.g., create or delete a task) in a general way. A “*Crew*” class stores the crew information, i.e., the role of the crew, amount available and salary of the crews.

Chapter 5. Java Application Design for Resource-Constrained Project Scheduling

Fig. 5.2 Domain class structure for plan and schedule package

5.3.2.2 GA Package

The “GA” Package describes the data structures required for the resource-constrained project scheduling genetic algorithms to select and recombine best project schedules. Figure 5.3 shows a detailed class diagram of the GA package and classes. The GA package consists of ten classes: *Chromosome*, *ChromString*, *GA*, *GAString*, *GASequenceList*, *ResourceScheduling*, *TopologicalSort*, *GAMultiobjective*, *TimeCostTradeOff*, and *GeneticAlgorithms*.

Chapter 5. Java Application Design for Resource-Constrained Project Scheduling

The *Chromosome* class is an abstract chromosome class which stores the genes. Class *ChromString* extends the abstract *Chromosome* class to store genetic codes in the form of strings. Using inheritance, much of the coding of the superclass is reused in the derived subclasses, avoiding duplication and benefiting from the shared data and functions among different classes.

The *GA* class is an abstract genetic algorithm class that contains chromosome objects as instance variables and implements the basic methods for doing crossover genetic mating, mutations, and iterating through the simulated evolution. Therefore it is able to implement most of the code in this *GA* ancestor class whose methods operate on abstract chromosomes. The abstract *GA* class is extended with one abstract subclass: *GAString* which works with *ChromString*. The abstract *GAString* class is further extended with two abstract subclasses: *GAMultiobjective* which contains additional methods for multiobjective problems; and *GASequenceList* which contains additional methods for sorting sequences and preventing duplicate genes in resource scheduling problems. These classes are abstract to allow users to implement abstract methods that are specific to a particular GA problem, such as initialization of chromosomes, mutation and crossover methods.

Class “*TopologicalSort*” provides functions to schedule tasks based on the precedence relationships between tasks and tasks’ priorities as described in Chapter 4. Class “*Genetic Algorithms*” is to initialize and run the genetic algorithms with functions like storing the initial values of the population size, number of generation, crossover rate and mutation rate, importing project data from “Plan” package and running the GA model.

Chapter 5. Java Application Design for Resource-Constrained Project Scheduling

Fig. 5.3 Domain class structure for GA package

The Unified Modeling Language (UML) component that most clearly describes how algorithm is to be preceded is the “Sequence Diagram.” The Sequence Diagram provides the graphical means to describe the control of processing within a given algorithm. Markers to describe object activation and iterations are included in the diagram to describe the flow control within object-oriented programs. Where needed, sub-sequence diagrams display iteration or conditional branching during a sequence. A sequence diagram for the use case Add Crew is shown in Figure 5.4.

Fig. 5.4 Sequence diagram for the Add Crew scenario

When modeling the sequence diagrams, windows and dialogs are needed to provide an interface to the actors. In this analysis, interface windows are needed for creating or

Chapter 5. Java Application Design for Resource-Constrained Project Scheduling

editing project, task and crew; displaying Gantt chart and task tree; and inputting values of GA parameters for the two-phase GA model.

To separate the windows classes in the analysis from the domain classes, the window classes are grouped into a package named “GUI Package”, and domain classes are grouped into a package named “Business Package”.

5.4 Design

The design phase expands and details the analysis model by taking into account all technical implications and restrictions. The purpose of the design is to specify a working solution that can be easily translated into programming code. The design can be divided into two segments:

- **Architectural design.** This is the high-level design where the packages are defined, including the dependencies and primary communication mechanisms between the packages.
- **Detailed design.** Here all classes are described in enough detail with clear specifications for coding.

5.4.1 Architecture Design

A software system consists of a number of packages. The packages can concern either handling of a specific functional area or a specific technical area. It is vital to separate the application logic (the domain classes) from the technical logic so that changes in either don't impact the other part. The architecture is the foundation for an extensible and changeable system. There are two purposes to conduct an architecture design. One goal is to identify and set up rules for dependencies between the packages so that no bidirectional dependencies are created between packages (in order to avoid packages becoming too tightly integrated with each other). Another goal is to identify the need for standard libraries. Libraries available today address technical areas such as the user

Chapter 5. Java Application Design for Resource-Constrained Project Scheduling

interface, the database, or communication, but more application-specific libraries are expected to emerge as well.

The packages in the project scheduling system are as follows:

- **User-Interface (UI) Package.** These classes are based on the Java AWT (Abstract Windows Toolkits) package, a standard library in Java for writing user-interface applications. This package cooperates with the Business-Objects package, which contains the classes where the data is actually stored. The UI package calls operations on the business objects to retrieve and insert data into them.
- **Business-Objects Package.** This includes the domain classes from the analysis model. The business-objects package consists of two sub-packages: “Plan” package and “GA” package. The design completely defines their operations and adds support for persistence. The business-object package cooperates with the database package in that all business-object classes must inherit from the *Persistent* class in the Database package.
- **Database Package.** The Database package supplies services to other classes in the Business-Object package so that they can be stored persistently. In the current version, the *Persistent* class will store objects of its subclasses to files in the file system.
- **Utility Package.** The Utility package contains services that are used in other packages in the system. Currently the *objId* class is the only one in the package. It is used to refer to persistent objects throughout the system including the User-Interface, Business-Object, and Database packages.

The design of these packages is shown in Figure 5.5.

Chapter 5. Java Application Design for Resource-Constrained Project Scheduling

Fig. 5.5 Architectural overview of project scheduling system

5.4.2 Detailed Design

The detailed design describes the new technical classes—the classes in the User-Interface and Database packages—and fleshes out the Business-Object classes sketched during the analysis. The class and dynamic diagrams used are the same diagrams used in the analysis, but they are defined on a more detailed and technical level.

5.4.2.1 Database Package

Chapter 5. Java Application Design for Resource-Constrained Project Scheduling

The project scheduling system must have objects stored persistently; therefore a database layer must be added to provide this service. For simplicity, the objects are stored as files on the disk. Details about the storage are hidden from the application, which calls common operations such as *store()*, *update()*, *delete()*, and *find()*. These are part of a class called *Persistent*, which all classes that need persistent objects must inherit.

An important factor in the persistence handling is the *ObjId* class, whose objects are used to refer to any persistent object in the system (regardless of whether the object is on disk or has been read into the application). *ObjId*, short for object identity, is a well-known technique for handling object references elegantly in an application. By using object identifiers, an object ID can be passed to the generic *Persistent.getObject()* operation and the object will be retrieved from persistent storage and returned. Usually this is done through a *getObject* operation in each persistent class, which also performs necessary type checks and conversions. An object identifier can also be passed easily as a parameter between operations (e.g., a search window that looks for a specific object can pass its result to another window through an object ID).

The *ObjId* is a general class used by all packages except the “GA” package in the system (User Interface, Business Objects, and Database) so it has been placed in a Utility package in the design rather than in the Database package.

5.4.2.2 Business-Objects Package

The Business-Objects package in the design is based on the domain classes in the corresponding packages in the analysis. The classes, their relationships, and behavior are preserved, but the classes are described in more detail, including how their relationships and behavior are implemented. The class diagrams for “Plan” sub-package and “GA” sub-package in business-object package in detailed design stage are plotted in Figure 5.6 and Figure 5.7 respectively. These diagrams flesh out the detailed design of the various classes in the business-objects package.

Chapter 5. Java Application Design for Resource-Constrained Project Scheduling

Fig. 5.6 Detailed design for “Plan” sub-package

Chapter 5. Java Application Design for Resource-Constrained Project Scheduling

Fig. 5.7 Detailed design for "GA" sub-package

5.4.2.3 User-Interface Package

The User-Interface package presents the services and information in the system to a user. As noted, this package is based on the standard Java AWT (Abstract Window Toolkit) class. Figure 5.8 shows the class diagram in the user-interface package. This contains typical AWT event handlers. The attributes for buttons, labels, and edit fields are not shown. The classes in the user-interface package all are one-to-one associations.

Chapter 5. Java Application Design for Resource-Constrained Project Scheduling

Fig. 5.8 Class diagram in user-interface package

The dynamic models in the design model have been allocated to the GUI package, since all interactions with the user are initiated through the user interface. Again, sequence diagrams have been chosen to show the dynamic models. The design model's realizations of the use cases are shown in exact detail, including the actual operations on the classes.

The sequence diagrams are actually created in a series of iterations. Discoveries made in the implementation (coding) phase result in further iterations. Figure 5.9 shows the resulting design sequence diagram for Add Task.

Chapter 5. Java Application Design for Resource-Constrained Project Scheduling

Fig. 5.9 Sequence diagram for Add Task

5.5 Implementation

Programming begins during implementation phase. The requirements for this application specify that the system be able to run on a number of different processors and operating systems, so Java was chosen to implement the system. Java makes mapping the logical classes to the code components easy, because there is a one-to one mapping of a class to a Java code file. The Java codes for the core part of the application, i.e., the GA package are in Appendix for reference.

Chapter 5. Java Application Design for Resource-Constrained Project Scheduling

5.6 Demonstration

Figure 5.10 shows the initial window encountered by the project engineers.

Fig. 5.10 Main window of the application

5.6.1 Menu Items

Figure 5.11 shows the menu items of the application.

Fig. 5.11 Menu items of the application

Chapter 5. Java Application Design for Resource-Constrained Project Scheduling

The following is the description of the menu items:

Project: There are six sub-menus under this menu:

1. **New:** create a new project
2. **Open:** open a project file that contains saved project information
3. **Save:** save the project into an XML file
4. **Save As:** save the project into an XML file with other name as shown in Figure 5.12. XML is a markup language for documents containing structured information. It can be used to store any kind of structured information, and to enclose or encapsulate information in order to pass it between different computing systems which would otherwise be unable to communicate.
5. **Properties:** allow user to input the general project information, i.e., the project title, brief description of the project, the owner of the project and the indirect cost rate, as shown in Figure 5.13
6. **Quit:** quit from the program. If there is any unsaved information when quitting, a warning dialog will pop up to remind the user to save the project before quitting.

Fig. 5.12 Saving project as an XML file

Chapter 5. Java Application Design for Resource-Constrained Project Scheduling

Fig. 5.13 Project frame of the application

Task: There are five sub-menus under this menu:

1. **New Task:** create a new task as shown in Figure 5.14. The user can input the task-related information, i.e., the task id, task name, the percentage of work completed, interruptions cost if interruptable, start date, and corresponding duration, crew requirements and overlap days for each execution mode. The user has to select one and only one mode to display the task in the Gantt chart panel.
2. **Delete Task:** delete the task from the project
3. **Task Properties:** view and update the task information as shown in Figure 5.15. The user can define the precedence relationships between tasks in this frame.
4. **Edit Notes:** add text format notes to the task as shown in Figure 5.16
5. **Genetic Algorithms:** as shown in Figure 5.17, it pops up a frame for users to input necessary GA related information, such as population size, generation, crossover rate and mutation rate for time-cost trade-off subsystem and resource scheduling subsystem in the GA model. Users can start GA optimization when all the required data are keyed in.

Chapter 5. Java Application Design for Resource-Constrained Project Scheduling

The 'New task' dialog box contains the following fields and controls:

- Task Id:** 3
- Name:** Task 3
- Progress:** 0
- Interruptable:** ☒
- Interruption Cost:** 100
- Begin date:** 1/1/2014
- task information table:**

Executionmo...	Duration(day)	Labor	Engineer	Overlap(day)	Display
1	6	13	0	0	☒
2	8	9	0	0	☐
3	10	7	0	0	☐
					☐
					☐
					☐

Buttons:

Fig. 5.14 Task frame for creating new task

The 'Properties for Task 1' dialog box contains the following fields and controls:

- Task Id:** 1
- Name:** Task 1
- Progress:** 0
- Interruptable:** ☒
- Interruption Cost:** 100
- Begin date:** 1/1/2014
- Successors:** none, Task 2, Task 3, Task 4, Task 5, Task 6, Task 7, Task 8
- task information table:**

Executionmo...	Duration(day)	Labor	Engineer	Overlap(day)	Display
1	5	15	0	0	☒
					☐
					☐
					☐
					☐

Buttons:

Fig. 5.15 Task frame for viewing task property

The 'Notes for Task 1' dialog box contains the following fields and controls:

- Notes:** This is the first task of the project

Buttons:

Fig. 5.16 Task notes of the application

Chapter 5. Java Application Design for Resource-Constrained Project Scheduling

Fig. 5.17 GA frame of the application

Resource: There are three sub- menus under this menu:

- 1. **New Resource:** create new crew as shown in Figure 5.18. Users can input crew-related information, such as the functions of the crew, the amount available and the unit salary of the crew.
- 2. **Delete Resource:** delete the crew from the project
- 3. **Resource Properties:** view and update crew information

Fig. 5.18 Crew frame of the application

5.6.2 Gantt Chart

To test the two-phase GA model built in the application, a simple project is planned with the network shown in Figure 4.20 and activity data in Table 4.2.

Chapter 5. Java Application Design for Resource-Constrained Project Scheduling

5.6.2.1 Precedence Relationships Only

If only precedence relationships are active, the Gantt chart for optimal project duration, which is determined from the GA optimization (see Section 4.4.1), is depicted in Figure 5.19.

Fig. 5.19 Gantt chart for optimal project duration of 49 days under precedence constraint

5.6.2.2 Adding Resource Capacities

If both precedence relationships and resource capacities are selected as the active constraints, the Gantt chart for optimal project duration, which is obtained from GA optimization (see Section 4.4.2), is depicted in Figure 5.20.

Chapter 5. Java Application Design for Resource-Constrained Project Scheduling

Fig. 5.20 Gantt chart for optimal project duration of 56 days under precedence and resource constraints

5.6.2.3 Adding All Project Constraints

If all the project constraints are selected, i.e., the precedence relationships, resource capacities, interruption for ongoing activities and overlap of selected activities, the Gantt chart for optimal project duration, which is determined from the GA optimization (see Section 4.4.3), is depicted in Figure 5.22. We assume that only activity 1 and 3 are started when the project has been commenced for one day. Activity 4, 8, and 9 allow overlap by their successor activity 10 by one day. The Gantt chart at current stage is shown in Figure 5.21. In Figure 5.22, it can be seen that the ongoing activity 1 was interrupted and activity 10 overlaps its predecessors by one day.

Chapter 5. Java Application Design for Resource-Constrained Project Scheduling

Fig. 5.21 The Gantt chart at current stage before re-optimization

Fig. 5.22 Gantt chart for project duration of 56 days under all project constraints

5.7 Summary

An object-oriented Java application for resource-constrained project scheduling has been described in this chapter. The application is developed in four main stages: requirement, analysis, design and implementation. The requirement stage produces a requirements specification that defines what the application will do. The practical requirements in project scheduling management include: (1) Creation of a project plan; (2) optimization

Chapter 5. Java Application Design for Resource-Constrained Project Scheduling

of schedule through time-cost analysis; (3) application of resource scheduling to reflecting the real construction practice. The analysis stage provides an outline of the model classes and their interactions. The design stage provides a detailed design of the classes to satisfy the practical requirements in project scheduling management. The implementation stage maps the object-oriented model to Java code to produce a window based application that runs on Microsoft XP with user-friendly interfaces. The demonstration shows the application work as expected.

CHAPTER 6 CONCLUSIONS

6.1 Thesis Review

This research focuses on the development of a two-phase genetic algorithm model for resource-constrained project scheduling problems and an object-oriented (OO) application incorporating the proposed GA model to support construction project scheduling management.

The literature pertaining to this research is described and evaluated in Chapter 2. The characterization of resource-constrained project scheduling problem was reviewed, from which time-cost trade-off analyses and resource scheduling were identified as key issues to solve resource-constrained project scheduling problems. Then research previously accomplished in the area of resource-constrained project scheduling, emphasizing on time-cost trade-off modeling and resource scheduling, were described and areas of opportunity for new research were identified. Object-oriented modeling techniques were reviewed for the development of an object-oriented model for resource-constrained project scheduling.

A comprehensive description of genetic algorithms is given in the Chapter 3. One major objective of the work is to propose a genetic algorithm methodology to solve the resource-constrained project scheduling problem. The resource-constrained project scheduling problems can be treated as a combination of multiple objective problems (time-cost trade-off) and resource scheduling problems. Hence, the multiple objective problems and resource scheduling problems were analyzed. In this research, it is attempted to solve the resource-constrained project scheduling problems by integrating the two different genetic algorithms (the multiobjective genetic algorithms and the resource-constrained genetic algorithms). In brief, the genetic algorithm description and

Chapter 6. Conclusions

the computer system built on that algorithm in the following chapters derived from the outcome of the investigation of genetic algorithms.

Chapter 4 has presented the characteristics of resource-constrained project scheduling problems and proposed a two-phase GAs model for resource-constrained project scheduling, which incorporated GA-based computational techniques for time-cost trade-off and resource scheduling. A set of constraints are provided in the model for project scheduling problems. If only precedence relationships are selected, the time-cost trade-off result is comparable to the work done by Feng et al. (1997). On the other hand, if more project constraints are applied, a corresponding practical time-cost trade-off result can be generated using this model. Allocations of limited resources, interruption and/or overlap of activities are allowed in the scheduling process.

An object-oriented Java application for resource-constrained project scheduling has been described in Chapter 5. The application is developed in four main stages: requirement, analysis, design and implementation. The requirement stage produces a requirements specification that defines what the application will do. The practical requirements in project scheduling management include: (1) Creation of a project plan; (2) optimization of schedule through time-cost analysis; (3) application of resource scheduling to reflect the real construction practice. The analysis stage provides an outline of the model classes and their interactions. The design stage provides a detailed design of the classes to satisfy the practical requirements in project scheduling management. The implementation stage maps the object-oriented model to Java code to produce a window-based application that runs on Microsoft XP with user-friendly interfaces. The demonstration shows the application work as expected.

6.2 Contributions

In general, this research contributes to the field of construction project scheduling management. The specific contributions are as follows:

Chapter 6. Conclusions

- **A proposed two-phase GA model for resource-constrained project scheduling problems:** A two-phase GA (genetic algorithms) model is proposed, in which both the effects of time-cost trade-off and resource scheduling are taken into account. A GA-based time-cost trade-off analysis is adopted to select the execution mode of each activity through the balance of time and cost, followed by utilization of a GA-based resource scheduling method to generate a feasible schedule which may satisfy all the project constraints.
- **An object-oriented software system for resource-constrained project scheduling:** A software system is developed based on the object-oriented information model and the proposed two-phase GA model.

Thus, the research develops an object-oriented resource-constrained project scheduling system that can improve the efficiency in resolving real construction project scheduling problems.

6.4 Future Research

In the future, the research work could include:

- **Extension to multi-resource driven scheduling and building a resource data base linking to the system:** Currently the resource scheduling is limited to labor resource scheduling only. In the construction practice, the resources include labor, material and equipment. An activity normally will use more than one kind of resource. Hence, it is desired to extend the single resource driven scheduling to multi-resource driven scheduling. A well-constructed resource database is needed to store the resource information by the system as well.
- **Improvement to Graphic User Interface:** The time-cost trade-off plot generated by the application only gives a brief view to the time-cost optimization, more functions should be added, such as the indication of the duration and cost and plot of the resource profile for each solution on the time-cost trade-off plot. Furthermore, the interrupted ongoing activity should be split into two separated

Chapter 6. Conclusions

sub-tasks instead of an unbroken one as shown in Figure 5.25. The critical path should be indicated in the Gantt chart to provide more information to the user.

- **Extension of the system with the ability to exchange information with other IFC (Industrial Foundation Class) compatible software:** The system can hold the promise to improve project scheduling management practice, but it requires extensive underlying technical foundations, particularly standardized data models to enable information sharing among computer applications. The International Alliance for Interoperability (IAI) is developing Industry Foundation Classes (IFC) to provide support for all architecture, engineering, construction, and facilities management industries. In addition to physical information about buildings, these classes represent project management information such as estimating and scheduling data. Many core concepts relating to cost and scheduling portions of the Industry Foundation Classes have been added to the latest Release IFC2x, making IFC realistic to represent the estimating and scheduling processes. The system will be designed with the compliance to the IFC Release 2x, hence the system can import and export IFC format files to achieve the goal of sharing information with other IFC compatible application software.

References

REFERENCES

- Adeli, H., and Karim, A. (1997). "Scheduling/cost optimization and neural dynamics model for construction." *J. Constr. Engrg. and Mgmt.*, ASCE, 123(4), 450-458.
- Badiru, A. B. (1991). *Project management tools for engineering management professionals*, Georgia, USA: Norcross.
- Bell, C., and J. Han. (1991). "A new heuristic solution method in resource constrained project scheduling." *Naval Research Logistics*, vol.38, 315-331.
- Bent, J. A., and Thuman, A. (1989). *Project Management for engineering and construction*, New York: Fairmont Press.
- Bowers, J. A. (1995). "Criticality in resource constrained networks." *Journal of the Operational Research Society*, 46(1), 80-91.
- Brand, J. D., Meyer, W. L., and Shaffer, L. R. (1964). "The Resource Scheduling problem in Construction." *Construction Research Series (2)*, Department of Civil Engineering, University of Illinois.
- Chan, W. T., Chua, D. K. H., and Kannan, G., (1996). "Construction resource scheduling with genetic algorithms." *J. Constr. Eng. Manage.*, ASCE, 122(2), 125-132.
- Coad, P., and Yourdon, E. (1991). *Object-Oriented Analysis*, Prentice-Hall, NJ.
- Cohen, G., and Levitt, R. E. (1991). "An object-oriented model of information sharing in project design teams." *Proceedings of Construction Congress'91, Preparing for Construction in the 21st Century*, ASCE, Cambridge, MA, 348-353.

References

- Cohenca, D., Laufer, A., and Ledbetter, W. B. (1989). "Factors affecting construction planning efforts." *J. Constr. Eng. Manage.*, 115(1), 70-87.
- Cohon, J. L. (1978). *Multiobjective Programming and Planning*. New York: Academic Press.
- Cohon, J. L. (1985). "Multicriteria programming: brief review and application." In J. S. Gero (Ed.), *Design Optimization*, 163–191. Orlando, Florida: Academic Press.
- Davis, E. W., and Patterson, J. H. (1975). "A Comparison of Heuristic and Optimum Solutions in Resource-Constrained Project Scheduling." *Manage. Sci.*, 21(8), 944-955.
- Davis, L. (1985). "Job Shop Scheduling with Genetic Algorithms." *Proceedings of an International Conference on Genetic Algorithms and their Applications*, Pittsburgh, Lawrence Erlbaum Associates.
- Davis, L. (1991). *A handbook of genetic algorithms*. Van Nostrand Reinhold, New York, N.Y.
- De Jong, K. A. (1975). *An analysis of the behavior of a class of genetic adaptive systems*. Ph. D. thesis, University of Michigan.
- De, P., Dunne, E.J., and Wells, C.E. (1995). "The discrete time-cost tradeoff problem revisited." *Euro. J. Operational Res.*, 81, 225-238.
- Deb, K. (1999). "Evolutionary algorithms for multi-criterion optimization in engineering design." In *Proceedings of Evolutionary Algorithms in Engineering and Computer Science (EUROGEN'99)*.
- DeJong, D. A., Spears, and William, M. (1989). "Using Genetic Algorithms to Solve NP-Complete Problems." *International Conference on Genetic Algorithms*, 124-132.

References

- Drexel, A., and Gruenewald, J. (1993). "Nonpreemptive multi-mode resource constrained project scheduling." *IIE Transaction*, vol 25, 74-81.
- Dyer, James S., Fishburn, Peter C., Steuer, Ralph E., Wallenius, Jyrki, Zionts, Stanley. (1992). "Multiple Criteria Decision Making, Multiattribute Utility Theory: The Next Ten Years," *Management Science*, 38(5), 645-666.
- Elmagraby, S. E. (1993). "Resource allocation via dynamic programming in activity networks." *Euro. J. Operational Res.*, 64, 199-215.
- Feng, C., Liu, L., and Burns, S. (1997). "Using genetic algorithms to solve construction time-cost trade-off problems." *J. Comp. Civ. Engrg.*, ASCE, 11(3), 184-189.
- Fondahl, J. W. (1961). "A non-computer approach to the critical path method for the construction industry." *Tech. Rep. No. 9*, The Constr. Inst., Dept. of Civ. Engrg., Stanford Univ.
- Fonseca, C. M., and Fleming, P. J. (1993). "Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization." In S. Forrest (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms*, San Mateo, California, 416-423. Morgan Kaufmann.
- Fowler, M., and Scott, K. (1997). *UML distilled: Applying the standard object modeling language*. Addison-Wesley Longman, Reading, Mass.
- Gen, M. and Cheng, R. (1997). *Genetic Algorithms & Engineering Design*, John Wiley & Sons, Inc. New York, N.Y.
- Goldberg, D. E., and Richardson J. (1987). "Genetic algorithms with sharing for multimodal function optimization." In J. J. Grefenstette (Ed.), *Genetic Algorithms*

References

- and their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, Hillsdale, NJ, 41–49. Lawrence Erlbaum.
- Goldberg, D. (1989). *Genetic Algorithm in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA.
- Gong, D., and Rowings, Jr. J.E. (1995). "Calculation of safe float use in risk-analysis-oriented network scheduling." *Int. J Project Management*, 13(3), 187-194.
- Hackman, S.T., and Leachman, R.C. (1989). "A General Framework for Modeling Production." *Management Science*, 35(4), 478-495.
- Hajela, P., and Lin, C. Y. (1992). "Genetic search strategies in multicriterion optimal design." *Structural Optimization* 4, 99–107.
- Henderickson, C., and Au, T. (1989). *Project management for Construction*. Prentice-Hall, Inc., Englewood Cliffs, N.J.
- Holland, J. H. (1975). *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor, Mich.
- Horn, J., Nafpliotis, N., and Goldberg D. E. (1994). "A niched Pareto genetic algorithm for multiobjective optimization." In *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Computation*, Volume 1, Piscataway, NJ, 82–87. IEEE.
- Ishibuchi, H., and Murata, T. (1996). "Multi-objective genetic local search algorithm." In *Proceedings of 1996 IEEE International Conference on Evolutionary Computation (ICEC'96)*, Piscataway, NJ, 119–124. IEEE.

References

- Karim, A., and Adeli, H. (1999). "OO information model for construction project management." *J. Constr. Eng. Manage.*, ASCE, 125(5) 361–367.
- Kelly, James E. Jr. (1961). "Critical path planning and scheduling: mathematical basis." *Operations Res.*, 9(3), 167-179.
- Kolisch, R., Sprecher, A., and Drexl, A. (1992). "Characterization and Generation of a General Class of Resource-Constrained Project Scheduling Problems." Institut f.r Betriebswirtschaftslehre, Universit.t zu Kiel.
- Koski, J. (1984). "Multicriterion optimization in structural design." In E. Atrek, R. H. Gallagher, K. M. Ragsdell, and O. C. Zienkiewicz (Eds.), *New Directions in Optimum Structural Design*, 483–503. Wiley.
- Knowles, J. D., Watson, R. A., and Corne, D. W. (2001). "Reducing local optima in single-objective problems by multi-objectivization." *EMO 2001*, E. Zitzler et al., eds., LNCS 1993, 269–283.
- Li, H., and Love, P. (1997). "Using improved genetic algorithms to facilitate time-cost optimization." *J. Constr. Eng. Manage.*, 123(3), 233–237.
- Leu, S. S., Chen, A. T., and Yang, C. H. (2001). "A GA-based fuzzy optimal model for construction time-cost trade-off." *Int. J. Pro. Mgmt.*, 19(1), 47-58
- Matthews, M. (1994). "Resource scheduling: incorporating capacity into schedule construction." *Project Management Journal*, 25(2), 44-54.
- Meyer, W. L., and Shaffer, L. R. (1963). "Extensions of the Critical path Method through the Application of Integer Programming." Construction Research Series (2), Department of Civil Engineering, University of Illinois.

References

- McCough, E.H. (1982). "Scheduling: effective methods and techniques." *J. Constr. Div*, 108(1), 75-84.
- Moselhi, O. (1993). "Schedule compression using the direct stiffness method." *Can. J. Civ. Engrg.*, 20, 65-72.
- Ng, S. T., Deng, M. Z. M., Skitmore, R. M., and Lam, K. C. (2000). "A conceptual case-based decision model for mitigating construction delays." *Int. J. Constr. Inf. Technol.*, 8(2), 1-20.
- Nkasu, M. M. (1994). "COMSARS: a computer-sequencing approach to multiresourceconstrained scheduling." *Int. J. of Project Management*, 12(3), 183-192.
- Pagnoni, A. (1990). *Project engineering: computer oriented planning and operational decision making*. Springer-Verlag KG, Berlin.
- Panwalkar, S. S., and Iskander, W. (1977). "A Survey of Scheduling Rules." *Oper. Res.*, 25(1): 45-61.
- Pareto, V. (1896). *Cours D'Economie Politique*, Volume 1. Lausanne: F.Rouge.
- Paulson, B.C. (1973). "Project planning and scheduling: unified approach." *J. Constr. Div*, 99(1), 45-58.
- Prager, W. (1963). "A structural method of computing project cost polygons." *Mgmt. Sci.* 9(3), 394-404.
- Pritsker, A., Waters, L., and Wolfe, P. (1969). "Multiproject scheduling with limited resources: a 0-1 approach." *Mgmt. Sci.*, 16, 93-108.

References

- Ravindran, A., Phillips, D. T., and Solberg, J. J. (1987). *Operations Research: Principals and Practice*, 2nd ed., Wiley & Sons, New York, pg. 472.
- Ressel, A. D., and Caselton, W. F. (1988). "Extensions to linear scheduling optimization." *J. Constr. Engrg. and Mgmt.*, ASCE, 114(1), 36-52.
- Robinson, D. R. (1975). "A dynamic programming solution to cost-time tradeoff for CPM." *Mgmt. Sci.*, 22(2), 158-166.
- Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W. (1991). *Object-oriented modeling and design*, Prentice-Hall, Englewood Cliffs, N.J.
- Sampson, S., and Weiss, E. (1993). "Local search techniques for the generalized resource constrained project scheduling problem." *Naval Research Logistics*, vol 40, 665-675.
- Satyanarayana, K., Rajeev, S., Kalidindi, S., and Kalyanaraman, V. (1993). "Optimum resource allocation in construction projects using genetic algorithms." *Proc., 3rd Int. Conf. on the Application of AI to Civil and Structural Engineering*, Edinburgh, U.K., 17-19.
- Shi, J., and Deng, Z. M. (2000) "Object-oriented resource-based planning method (ORPM) for construction." *Int. J. Pro. Mgmt.*, 18(3), 179-188.
- Siemens, N. (1971). "A simple CPM time-cost trade-off algorithm." *Mgmt. Sci.*, 17(6), 354-363.
- Sprecher, A. (1994). "Resource-Constrained Project Scheduling: exact methods for the multi-mode case." *Lecture Notes in Economics and Mathematical Systems* 409.
- Srinivas, N. and Deb, K. (1994). "Multiobjective optimization using nondominated sorting in genetic algorithms." *Evolutionary Computation* 2(3), 221-248.

References

- Steuer, R. E. (1986). *Multiple Criteria Optimization: Theory, Computation, and Application*. New York: Wiley.
- Syswerda, G. (1991). "Schedule optimization using genetic algorithms." *Handbook of genetic algorithms*, L. Davis, ed., Van Nostrand Reinhold, New York, N.Y.
- Tamaki, H., Mori, M., Araki, M., Mishima, Y., and Ogai, H. (1994). "Multicriteria optimization by genetic algorithms: A case of scheduling in hot rolling process." In M. Fushimi and K. Tone (Eds.), *Proceedings of APORS'94*, Singapore, 374–381. World Scientific Publishing.
- Veldhuizen, D. A. V. (1999). *Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations*. Ph. D. thesis, Graduate School of Engineering of the Air Force Institute of Technology, Air University.

Appendix-Java Source Codes for GA Package

APPENDIX-
JAVA SOURCE CODES FOR GA PACAKAGE

This appendix contains the JAVA source codes for the GA package that make up the proposed two-phase GA model.

List of Classes

Chromosome.java	116
ChromString.java	116
GA.java	117
GAStrng.java	123
GASequenceList.java	124
ResouceScheduling.java	127
GAMultiobjective.java	129
TimeCostTradeOff.java	132
TopologicalSort.java	137
GeneticAlgorithms.java.....	143

Appendix-Java Source Codes for GA Package

Chromosome.java

```
package ga;
public abstract class Chromosome
{
    protected double fitness;
    abstract String getGenesAsStr();
    abstract void copyChromGenes(Chromosome chromosome);
}
```

ChromString.java

```
package ga;

/** chromosome class that stores genes as strings */
public class ChromString extends Chromosome
{
    protected char[] genes;

    /** return the array of genes as a string */
    public String getGenesAsStr()
    {
        String sGenes = "";
        for (int i = 0; i < genes.length; i++) sGenes += genes[i];
        return(sGenes);
    }

    /** return the gene indexed by iGene as a char */
    public char getGene(int iGene)
    {
        return(this.genes[iGene]);
    }

    /** set this chromosomes genes (array) to the given chromosome string */
    public void setGenesFromStr(String sChromosome)
    {
        for (int i = 0; i < genes.length; i++)
            this.genes[i] = sChromosome.charAt(i);
    }

    /** copy the genes from the given chromosome over the existing genes */
    public void copyChromGenes(Chromosome chromosome)
    {
        int iGene;

        ChromString chromString = (ChromString)chromosome;
        for (iGene = 0; iGene < genes.length; iGene++)
            this.genes[iGene] = chromString.genes[iGene];
    }

    //constructor
    public ChromString(int iGenesDim)
    {
        genes = new char[iGenesDim];
    }
}
```


Appendix-Java Source Codes for GA Package

}

GA.java

```

package ga;
import java.util.*;

public abstract class GA
{
    double mutationProb;           //probability of a mutation occurs during genetic
                                   // mating. For example, 0.03 means 3% chance
    protected int maxGenerations;   //maximum generations to evolve

    int randomSelectionChance; //1-100 (e.g. 10 = 10% chance of random selection--not based on
                                   //fitness). Setting nonzero randomSelectionChance helps maintain
                                   //genetic diversity during evolution
    double crossoverProb;          //probability that a crossover will occur during genetic mating
    protected int chromosomeDim;   //dimension of chromosome (number of genes)
    protected int populationDim;   //number of chromosomes to evolve. A larger population dim will
                                   //result in a better evolution but will slow the process down
    Chromosome[] chromosomes;      //storage for pool of chromosomes for current generation
    Chromosome[] chromNextGen;     //storage for temporary holding pool for next generation
                                   //chromosomes

    int[] bestFitnessChromIndexes; //indexes of fittest chromosome in current generation
    int[] worstFitnessChromIndexes; //indexes of least fit chromosome in current generation
    protected double[] fitnessArray; // The array to store fitness value for each chromosome

    protected double bestfitness;
    protected double worsefitness;

    protected ActivityInitialization init;
    abstract protected void initPopulation();
    abstract protected void doRandomMutation(int iChromIndex);
    abstract protected void doOnePtCrossover(Chromosome Chrom1, Chromosome Chrom2);
    abstract protected void doTwoPtCrossover(Chromosome Chrom1, Chromosome Chrom2);
    abstract protected void doUniformCrossover(Chromosome Chrom1, Chromosome Chrom2);
    abstract protected void calFitness();
    abstract protected int SetChromosomeDim();

    public GA(
        int populationDim,
        double crossoverProb,
        int randomSelectionChance,
        int maxGenerations,
        double mutationProb,
        ActivityInitialization init)
    {
        this.randomSelectionChance = randomSelectionChance;

        this.init=init;
        this.chromosomeDim=SetChromosomeDim();
    }

```

Appendix-Java Source Codes for GA Package

```

    this.populationDim = populationDim;

    this.chromosomes = new Chromosome[populationDim];
    this.chromNextGen = new Chromosome[populationDim];
    this.fitnessArray=new double[populationDim];

    this.crossoverProb = crossoverProb;
    this.maxGenerations = maxGenerations;
    this.mutationProb = mutationProb;

}

public double getMutationProb()
{
    return mutationProb;
}

public int getMaxGenerations()
{
    return maxGenerations;
}

public int getRandomSelectionChance()
{
    return randomSelectionChance;
}

public double getCrossoverProb()
{
    return crossoverProb;
}

public int getChromosomeDim()
{
    return chromosomeDim;
}

public int getPopulationDim()
{
    return populationDim;
}

public String[] getFittestChromosomes()
{
    String[] fittestChromosomes=new String[bestFitnessChromIndexes.length];
    for(int i=0; i<bestFitnessChromIndexes.length;i++){
        fittestChromosomes[i]=this.chromosomes[bestFitnessChromIndexes[i]].getGenesAsStr();
    }
    return fittestChromosomes;
}

public double[] getFittestChromosomesFitnesses()
{
    double[] fittestChromosomesFitnesses=new double[bestFitnessChromIndexes.length];
    for(int i=0; i<bestFitnessChromIndexes.length;i++){

```

Appendix-Java Source Codes for GA Package

```

        fittestChromosomesFitnesses[i]=this.chromosomes[bestFitnessChromIndexes[i]].fitness;
    }
    return fittestChromosomesFitnesses;
}

public double getOneFittestChromosomeFitness()
{
    double oneFittestChromosomeFitness
        =this.chromosomes[this.bestFitnessChromIndexes[0]].fitness;
    return oneFittestChromosomeFitness;
}

public int getOneFittestChromosomeIndex()
{
    return bestFitnessChromIndexes[0];
}

public double getFitness(int iChromIndex)
{
    return fitnessArray[iChromIndex];
}

/** return a integer random number between 0(inclusive) and upperBound(exclusive) */
int getRandom(int upperBound)
{
    int iRandom = (int)(Math.random() * upperBound);
    return(iRandom);
}

/** return a double random number between 0 (inclusive) and upperBound(exclusive) */
double getRandom(double upperBound)
{
    double dRandom = (Math.random() * upperBound);
    return(dRandom);
}

/**Do genetic evolution of this population of chromosomes.*/
public void evolve()
{
    calFitness();
    computeFitnessRankings();
    doGeneticMating();
    copyNextGenToThisGen();
}

public void finalResults()
{
    calFitness();
    computeFitnessRankings();
}

/**
    Select two parents from population, giving highly fit individuals a greater chance of being
    selected.
*/
public void selectTwoParents(int[] indexParents)

```

Appendix-Java Source Codes for GA Package

```

{
    int indexParent1 = indexParents[0];
    int indexParent2 = indexParents[1];
    boolean bFound = false;
    int index1, index2;

    while (bFound == false)
    {
        index1 = getRandom(populationDim); //get random member of population
        do{index2 = getRandom(populationDim);}while(index2==index1);
        if (randomSelectionChance > getRandom(100))
        {
            indexParent1 = index1;
            bFound = true;
        }
        else
        {
            //the greater a chromosome's fitness, the higher prob that it will be selected to reproduce
            if (this.chromosomes[index1].fitness < this.chromosomes[index2].fitness)
            {
                indexParent1 = index1;
                bFound = true;
            }
            else
            {
                indexParent1=index2;
                bFound=true;
            }
        }
    }

    bFound = false;
    while (bFound == false)
    {
        do{index1 = getRandom(populationDim);}while(index1==indexParent1);
        do{index2 = getRandom(populationDim);}while(index2==index1||index2==indexParent1);

        if (randomSelectionChance > getRandom(100))
        {
            indexParent2 = index1;
            bFound = true;
        }
        else
        {
            if (this.chromosomes[index1].fitness < this.chromosomes[index2].fitness)
            {
                indexParent2 = index1;
                bFound = true;
            }
            else
            {
                indexParent2=index2;
                bFound=true;
            }
        }
    }
}

```


Appendix-Java Source Codes for GA Package

```

    indexParents[0] = indexParent1;
    indexParents[1] = indexParent2;
}

void computeFitnessRankings()
{
    for (int i=0; i < populationDim; i++){
        this.chromosomes[i].fitness = getFitness(i);
    }

    int numBestFitnessChrom=0;
    int numWorstFitnessChrom=0;
    for(int i=0; i<populationDim;i++)//find the number of best and worst chrom
    {
        if(this.chromosomes[i].fitness==this.bestfitness){
            ++numBestFitnessChrom;
        }
        if(this.chromosomes[i].fitness==this.worsefitness){
            ++numWorstFitnessChrom;
        }
    }

    this.bestFitnessChromIndexes=new int[numBestFitnessChrom];
    this.worstFitnessChromIndexes=new int[numWorstFitnessChrom];

    int index1=0;
    int index2=0;

    for(int i=0; i<populationDim;i++)
    {
        if(this.chromosomes[i].fitness==this.bestfitness){
            this.bestFitnessChromIndexes[index1]=i;
            index1++;
        }
        if(this.chromosomes[i].fitness==this.worsefitness){
            this.worstFitnessChromIndexes[index2]=i;
            index2++;
        }
    }
}

/** Create the next generation of chromosomes by genetically mating fitter
 * individuals of the current generation. Also employ elitism (so the
 * fittest 2 chromosomes always survive to the next generation). This way
 * an extremely fit chromosome is never lost from our chromosome pool.
 */
void doGeneticMating()
{
    int iCnt, iRandom;
    int indexParent1 = -1, indexParent2 = -1;
    Chromosome Chrom1, Chrom2;

    iCnt = 0;

    //Elitism--fittest chromosomes automatically go on to next gen

```

Appendix-Java Source Codes for GA Package

```

    for(int i=0;i<bestFitnessChromIndexes.length;i++){

this.chromNextGen[iCnt].copyChromGenes(this.chromosomes[this.bestFitnessChromIndexes[i]]);
    iCnt++;
    }

    Chrom1 = new ChromString(chromosomeDim);
    Chrom2 = new ChromString(chromosomeDim);

do
{
    int indexes[] = { indexParent1, indexParent2 };
    selectTwoParents(indexes);
    indexParent1 = indexes[0];
    indexParent2 = indexes[1];

    Chrom1.copyChromGenes(this.chromosomes[indexParent1]);
    Chrom2.copyChromGenes(this.chromosomes[indexParent2]);

    if (getRandom(1.0) < crossoverProb) //do crossover
    {
        doCrossover(Chrom1, Chrom2);

        if(iCnt==populationDim)break;
        this.chromNextGen[iCnt].copyChromGenes(Chrom1);
        iCnt++;
        if(iCnt==populationDim)break;
        this.chromNextGen[iCnt].copyChromGenes(Chrom2);
        iCnt++;
    }
    else //if no crossover, copy parent chromosome "as is" into the offspring
    {
        // CREATE OFFSPRING ONE
        if(iCnt==populationDim)break;
        this.chromNextGen[iCnt].copyChromGenes(Chrom1);
        iCnt++;

        if(iCnt==populationDim)break;

        // CREATE OFFSPRING TWO
        this.chromNextGen[iCnt].copyChromGenes(Chrom2);
        iCnt++;
    }
}
while (iCnt < populationDim);
}

/** Copy the chromosomes previously created and stored in the "next"
 * generation into the main chromosome memory pool. Perform random
 * mutations where appropriate.
 */
void copyNextGenToThisGen()
{
    for (int i=0; i < populationDim; i++)
    {

```

Appendix-Java Source Codes for GA Package

```

        boolean isworst=false;

        this.chromosomes[i].copyChromGenes(this.chromNextGen[i]);

        for(int j=0; j<this.worstFitnessChromIndexes.length;j++){
            if(i==this.worstFitnessChromIndexes[j]&&this.bestfitness!=this.worsefitness){
                isworst=true;
                break;
            }
        }
        if(isworst){
            doRandomMutation(i);
            continue; //always mutate the chromosomes with the lowest fitness
        }

        if (getRandom(1.0) < mutationProb)
            doRandomMutation(i);
    }
}
}

```

GAStrng.java

```

package ga;

public abstract class GAStrng extends GA
{
    protected String possGeneValues;

    protected ChromString getChromosome(int index)
    {
        return((ChromString)this.chromosomes[index]);
    }

    public GAStrng(
        int populationDim,
        double crossoverProb,
        int randomSelectionChance,
        int maxGenerations,
        double mutationProb,
        String possGeneValues,
        ActivityInitialization init) throws GAException
    {
        super( populationDim, crossoverProb, randomSelectionChance,
            maxGenerations, mutationProb,init);

        if (possGeneValues.length() < 2)
            throw new GAException("There must be at least 2 possible gene values");

        this.possGeneValues = possGeneValues;

        //create the chromosomes for this population
        for (int i=0; i < populationDim; i++)
        {

```

Appendix-Java Source Codes for GA Package

```

        this.chromosomes[i] = new ChromString(chromosomeDim);
        this.chromNextGen[i] = new ChromString(chromosomeDim);
    }
}
}

```

GASequenceList.java

```

package ga;

import java.util.Vector;

public abstract class GASequenceList extends GAString
{
    protected double[] sequence;
    protected boolean[] ActivityInterruption;//size of activityNum, [0]=activity id 1, and so on...
    protected Vector FinishedActivity;
    protected Vector OngoingActivityIdMode;
    protected int ActivityNum;
    protected int ModeSum;
    protected int ongoingnotinterruption=0;

    public GASequenceList(
        int pPopulationDim,
        double pCrossoverProb,
        int pRandomSelectionChance,
        int pMaxGenerations,
        double pMutationProb,
        String pPossGeneValues,
        ActivityInitialization init) throws GAException
    {
        super(pPopulationDim, pCrossoverProb,
            pRandomSelectionChance, pMaxGenerations,pMutationProb, pPossGeneValues,init);

        if (pPossGeneValues.length() != chromosomeDim)
            throw new GAException("Number of Possible gene values must equal Chromosome
Dimension");
        sequence = new double[chromosomeDim];
        initPopulation();
    }
    /** create random chromosomes from the given gene space. */
    protected void initPopulation()
    {
        String sGene, sChromosome;
        String ActivityId;
        boolean interruptable=true;

        for(int i=0;i<ActivityNum;i++){// find the number of non-interruptable ongoing activities
            ActivityId=Integer.toString(i+1);
            if(OngoingActivityIdMode.contains(ActivityId)){
                interruptable=ActivityInterruption[i];
                if(!interruptable){ongoingnotinterruption++;}
            }
        }
    }
}

```


Appendix-Java Source Codes for GA Package

```

for (int i=0; i < populationDim; i++)
{
    sChromosome = "";
    for (int iGene=0; iGene < ActivityNum; iGene++)
    {
        ActivityId=Integer.toString(iGene+1);// activity id
        if(FinishedActivity.contains(ActivityId))continue;//exclude the finished activity
        if(OngoingActivityIdMode.contains(ActivityId))
        {
            interruptable=ActivityInterruption[iGene];
            if(!interruptable){//the ongoing activity which not allowing interruption is given highest priority
                do{
                    sGene="" + this.possGeneValues.charAt(this.possGeneValues.length()-
getRandom(ongoingnotinterruption)-1);
                }while(sChromosome.indexOf(sGene)>=0);
                sChromosome += sGene;
                continue;
            }
        }
        do
        {
            sGene = "" + this.possGeneValues.charAt(getRandom(this.possGeneValues.length()-
ongoingnotinterruption));
        }
        while (sChromosome.indexOf(sGene) >= 0);

        sChromosome += sGene;
    }
    ((ChromString)this.chromosomes[i]).setGenesFromStr(sChromosome);
}
}

// do mutation
protected void doRandomMutation(int iChromIndex)
{
    int iGene1, iGene2;
    char cTemp,cTemp2;

    do{
        iGene1 = getRandom(chromosomeDim);
        cTemp2=((ChromString)this.chromosomes[iChromIndex]).genes[iGene1];
    }while(this.possGeneValues.indexOf(cTemp2)>=(this.possGeneValues.length()-
ongoingnotinterruption));

    do{
        iGene2 = getRandom(chromosomeDim);
        cTemp2=((ChromString)this.chromosomes[iChromIndex]).genes[iGene1];
    }while(this.possGeneValues.indexOf(cTemp2)>=(this.possGeneValues.length()-
ongoingnotinterruption));

    cTemp = ((ChromString)this.chromosomes[iChromIndex]).genes[iGene1];
    ((ChromString)this.chromosomes[iChromIndex]).genes[iGene1] =
((ChromString)this.chromosomes[iChromIndex]).genes[iGene2];
    ((ChromString)this.chromosomes[iChromIndex]).genes[iGene2] = cTemp;
}

```

Appendix-Java Source Codes for GA Package

```

protected String getChromWithoutDuplicates(String sChromosome)
{
    int iPos;
    int iGeneLeftOut;
    String sGene, sGenesLeftOut, sRestOfChrom;

    //first get a string (a list) of all genes left OUT of this chrom
    sGenesLeftOut = "";
    for (int i=0; i < this.possGeneValues.length(); i++)
    {
        sGene = "" + this.possGeneValues.charAt(i);
        iPos = sChromosome.indexOf(sGene);
        if (iPos < 0) //this gene not found in chromosome
            sGenesLeftOut += sGene;
    }

    if (sGenesLeftOut.length() == 0) //no duplicate genes, so exit
        return(sChromosome);

    StringBuffer sbChromosome = new StringBuffer(sChromosome);
    StringBuffer sbGenesLeftOut = new StringBuffer(sGenesLeftOut);

    for (int i=0; i < chromosomeDim; i++)
    {
        sGene = "" + sbChromosome.charAt(i);
        sRestOfChrom = sbChromosome.substring(i+1, chromosomeDim);

        iPos = sRestOfChrom.indexOf(sGene);
        if (iPos > -1) //gene also found in a later part of the chromosome,
            //it is duplicated!
        {
            //assign duplicate gene a random value from the list of genes left out
            iGeneLeftOut = getRandom(sbGenesLeftOut.length());
            sbChromosome.setCharAt(iPos+i+1, sbGenesLeftOut.charAt(iGeneLeftOut));

            //take this "gene left out" out of the list (string) of available genes
            sbGenesLeftOut.deleteCharAt(iGeneLeftOut);
        }
    }

    return(sbChromosome.toString());
}

protected void doCrossover(Chromosome Chrom1, Chromosome Chrom2)
{
    String sNewChrom1, sNewChrom2;
    int iCrossoverPoint;
    String sChrom1, sChrom2;

    iCrossoverPoint = getRandom(chromosomeDim-2);
    sChrom1 = Chrom1.getGenesAsStr();
    sChrom2 = Chrom2.getGenesAsStr();

    // CREATE OFFSPRING ONE

```

Appendix-Java Source Codes for GA Package

```

    sNewChrom1 = sChrom1.substring(0, iCrossoverPoint) + sChrom2.substring(iCrossoverPoint,
chromosomeDim);

    // CREATE OFFSPRING TWO
    sNewChrom2 = sChrom2.substring(0, iCrossoverPoint) + sChrom1.substring(iCrossoverPoint,
chromosomeDim);

    ((ChromString)Chrom1).setGenesFromStr(sNewChrom1);
    ((ChromString)Chrom2).setGenesFromStr(sNewChrom2);

    String sChrom1a = getChromWithoutDuplicates(Chrom1.getGenesAsStr());
    String sChrom2a = getChromWithoutDuplicates(Chrom2.getGenesAsStr());
    ((ChromString)Chrom1).setGenesFromStr(sChrom1a);
    ((ChromString)Chrom2).setGenesFromStr(sChrom2a);
}

```

ResourceScheduling.java

```

package ga;

import java.util.*;

public class ResourceScheduling extends GASequenceList implements Runnable
{
    private InformationStore infostore;

    protected ActivityInitialization activityinit;
    protected int[] ExecutionMode;
    protected ActivityInfo activityinfo;
    protected double[] duration;
    protected double[] totalcost;
    protected int mainpop;
    protected int maingen;
    protected Vector startArray;//vector to hold startArray vector
    final String geneSpace="abcdefghijklmnopqrstuvwxyz";

    public ResourceScheduling(GaSetting set,InformationStore is,ActivityInitialization init,String
genevalue,int imainpop, int imaingen) throws GAException
    {
        super(
            set.getRsPop(),           //population has N chromosomes
            set.getRsCross(),         //crossover probability
            2,                        //random selection chance % (regardless of fitness)
            set.getRsGen(),           //max generations
            set.getRsMutation(),      //chromosome mutation prob.
            genevalue,               //gene space (possible gene values)
            init);
        infostore=is;
        mainpop=imainpop;
        maingen=imaingen;
        totalcost=new double[populationDim];
        duration=new double[populationDim];
        startArray=new Vector();
    }

```


Appendix-Java Source Codes for GA Package

```

}

/**Set the chromosome dim and exclude the finished activity*/
protected int SetChromosomeDim(){
    this.ActivityNum= this.init.GetActivityNum();//the total number of activity
    this.ModeSum=this.init.GetModeSum();//the total execution mode of the activity
    this.FinishedActivity=this.init.GetFinishedActivities();//the set of finished activity
    this.OngoingActivityIdMode=this.init.GetOngoingActivities();//the set of ongoing activity
    this.ActivityInterruption=this.init.GetActivityInterruption();//the set of activity interruption
    int chromdim=ActivityNum-FinishedActivity.size();
    return chromdim;
}

public void run(){
    /**Do genetic evolution of this population of chromosomes.*/
    int count=0;
    int imaingen=0;
    while(imaingen<maingen){

        int imainpop=0;
        while(imainpop<mainpop){
            ExecutionMode=infostore.Get();
            int iGen=0;
            while (iGen < (this.maxGenerations))
            {
                int iChrom=0;
                while(iChrom<this.populationDim){
                    calDurationCost(iChrom);
                    iChrom++;
                }
                if(iGen<this.maxGenerations-1)
                {startArray.clear();//only the last one is useful
                    evolve();}
                iGen++;
            }
            finalResults();

            infostore.putCostDuration(totalcost[getOneFittestChromosomeIndex()],duration[getOneFittestChromosomeIndex()]);
            infostore.putStartArray((Vector)startArray.elementAt(getOneFittestChromosomeIndex()));
            count++;
            imainpop++;
        }

        imaingen++;
    }
}

protected void calDurationCost(int iChromIndex)
{
    String sChromosome;

    sChromosome = this.getChromosome(iChromIndex).getGenesAsStr();

    int lenChromosome = sChromosome.length();
    int parray[]= new int[lenChromosome];

```


Appendix-Java Source Codes for GA Package

```

    for (int i=0; i < lenChromosome; i++)
    {
        int index=geneSpace.indexOf(sChromosome.charAt(i));
        parray[i]=index+1;
    }
    TopologicalSort topsort=new TopologicalSort(init,parray,ExecutionMode);
    topsort.sort();
    duration[iChromIndex]=topsort.GetDuration();//duration
    totalcost[iChromIndex]=topsort.GetTotalCost();//total cost correspondingly
    startArray.add(topsort.getStartDate());//startDate for chromosome i
}
public void calFitness(){
    double minduration;
    double maxduration;
    double totalsum=0;
    double durationtemp[]=new double[this.populationDim];
    double tempfit[]=new double[this.populationDim];
    for(int i=0;i<this.populationDim;i++){
        durationtemp[i]=duration[i];
        totalsum+=totalcost[i];
    }

    Arrays.sort(durationtemp);
    maxduration=durationtemp[this.populationDim-1];
    minduration=durationtemp[0];
    for(int i=0;i<this.populationDim;i++){
        fitnessArray[i]=(maxduration-duration[i]+0.5)/(maxduration-
minduration+0.5)+totalcost[i]/(maxduration-minduration+0.5)/totalsum;
    }

    for(int i=0;i<this.populationDim;i++){
        tempfit[i]=fitnessArray[i];
    }
    Arrays.sort(tempfit);
    this.bestfitness=tempfit[this.populationDim-1];
    this.worsefitness=tempfit[0]; //best may equal to worst, then all are best and worst
}
}

```

GAMultiObjective.java

```

package ga;
import java.util.Vector;
import java.util.Enumuration;

public abstract class GAMultiObjective extends GASTring{

    protected int activityNum;
    protected int[] activityChoice;
    protected Vector FinishedActivity;
    protected Vector OnGoingActivityIdMode;//("1",4,"2",2...)

    public GAMultiObjective(
        int pPopulationDim,

```

Appendix-Java Source Codes for GA Package

```

        double pCrossoverProb,
        int pRandomSelectionChance,
        int pMaxGenerations,
        double pMutationProb,
        String pPossGeneValues,
        ActivityInitialization init) throws GAException
    {
        super( pPopulationDim, pCrossoverProb,
            pRandomSelectionChance, pMaxGenerations,pMutationProb, pPossGeneValues,init);

        if (pPossGeneValues.length() != 2)
            throw new GAException("only 0 and 1 are the possible gene values");

        initPopulation();
    }

    protected void initPopulation(){

        for(int i=0; i<populationDim;i++){

            for(int iGene=0;iGene<chromosomeDim;iGene++){
                ((ChromString)this.chromosomes[i]).genes[iGene] = '0';
            }

            int c=0;
            for(int j=0;j<this.activityNum;j++)
            {
                // delete the finished activities
                if(FinishedActivity.contains(Integer.toString(j+1))){
                    continue;
                }
                // for ongoing activities
                if(OnGoingActivityIdMode.contains(Integer.toString(j+1))){
                    int index=OnGoingActivityIdMode.indexOf(Integer.toString(j+1));
                    int selecteditem=((Integer)OnGoingActivityIdMode.elementAt(index+1)).intValue();
                    ((ChromString)this.chromosomes[i]).genes[c+selecteditem-1] = '1';
                    c+=this.activityChoice[j];
                    continue;
                }
                int selectedIndex=0;
                if(i==0){
                    selectedIndex=0;
                }else if(i==1){
                    selectedIndex=this.activityChoice[j]-1;
                }else{
                    selectedIndex=getRandom(this.activityChoice[j]);
                }
                ((ChromString)this.chromosomes[i]).genes[c+selectedIndex] = '1';
                c+=this.activityChoice[j];
            }
        }
    }

    protected void doRandomMutation(int iChromIndex){

```

Appendix-Java Source Codes for GA Package

```

    int iGene0=0;
    int iGene=0;
    int temp=0;
    int temp2=0;
    //only mutate those unstarted activities,ongoing and completed are not considered
    do{
        temp=getRandom(this.activityNum);
    }while((FinishedActivity.contains(Integer.toString(temp+1))))|(OnGoingActivityIdMode.contains(Integer.toString(temp+1)));
    for(int i=0;i<=temp;i++){
        if(FinishedActivity.contains(Integer.toString(i+1)))continue;

        iGene0+=this.activityChoice[i];
    }

    do{
        temp2=getRandom(this.activityChoice[temp]);
    }while((((ChromString)this.chromosomes[iChromIndex]).genes[iGene0-temp2-1]=='1')&&(this.activityChoice[temp]!=1));
    iGene=iGene0-temp2-1;

    for(int j=(iGene0-this.activityChoice[temp]);j<iGene0;j++){
        ((ChromString)this.chromosomes[iChromIndex]).genes[j]='0';
    }
    ((ChromString)this.chromosomes[iChromIndex]).genes[iGene]='1';
}

protected void doCrossover(Chromosome Chrom1, Chromosome Chrom2)
{
    String sNewChrom1, sNewChrom2;
    int iCrossoverPoint=0;
    String sChrom1, sChrom2;
    int temp;

    do{
        temp=getRandom(this.activityNum);
    }while(FinishedActivity.contains(Integer.toString(temp+1)));//exclude the finished activities

    for(int i=0;i<=temp;i++){
        if(FinishedActivity.contains(Integer.toString(i+1)))continue;//exclude the finished activities
        iCrossoverPoint+=this.activityChoice[i];
    }

    sChrom1 = Chrom1.getGenesAsStr();
    sChrom2 = Chrom2.getGenesAsStr();

    // CREATE OFFSPRING ONE
    sNewChrom1 = sChrom1.substring(0, iCrossoverPoint) + sChrom2.substring(iCrossoverPoint, chromosomeDim);

    // CREATE OFFSPRING TWO
    sNewChrom2 = sChrom2.substring(0, iCrossoverPoint) + sChrom1.substring(iCrossoverPoint, chromosomeDim);

    ((ChromString)Chrom1).setGenesFromStr(sNewChrom1);
    ((ChromString)Chrom2).setGenesFromStr(sNewChrom2);
}

```

Appendix-Java Source Codes for GA Package

```
}
}
```

TimeCostTradeOff.java

```
package ga;

import java.util.*;

public class TimeCostTradeOff extends GAMultiObjective implements Runnable{

    private InformationStore infostore;
    protected int[] dominatenum;
    protected int[] nondominated;
    protected int[] ExecutionMode;
    protected double[] costduration;
    protected double[][]  icostduration;//cost &duration
    protected ActivityInitialization activityinit;
    protected Vector bestchroms;
    protected Vector startArray;

    public TimeCostTradeOff(GaSetting set,InformationStore is,ActivityInitialization init) throws
    GAEException{
        super(
            set.getTcPop(),    //300,50 //population has N chromosomes
            set.getTcCross(),    //crossover probability
            5,    //random selection chance % (regardless of fitness)
            set.getTcGen(),    //20,10    //max generations
            set.getTcMutation(),    //chromosome mutation prob.
            "01", //gene space (possible gene values
            init);
        infostore=is;
        icostduration=new double[this.populationDim][2];
        dominatenum=new int[this.populationDim];
        nondominated=new int[this.populationDim];
        costduration=new double[2];
        startArray=new Vector();
        bestchroms=new Vector();
    }

    protected int SetChromosomeDim(){

        this.activityNum=this.init.GetActivityNum();
        this.FinishedActivity=this.init.GetFinishedActivities();
        this.OnGoingActivityIdMode=this.init.GetOngoingActivities();
        this.activityChoice=this.init.GetActivityChoice();
        this.ExecutionMode=new int[activityNum];//for finished activity,mode=0

        int chrodim=0;
        for(int j=0;j<activityChoice.length;j++){
            chrodim+=activityChoice[j];
        }
        Enumeration enum=FinishedActivity.elements();
        while(enum.hasMoreElements()){
```


Appendix-Java Source Codes for GA Package

```

        String index=(String)enum.nextElement();
        int FinishedActivityIndex=Integer.parseInt(index);
        chrodim=activityChoice[FinishedActivityIndex-1];
    }
    return chrodim;
}

public void run(){
    /**Do genetic evolution of this population of chromosomes.*/
    int iGen=0;
    while (iGen < (this.maxGenerations))
    {
        System.out.println("MainLoop "+(iGen+1));
        int iChrom=0;
        while(iChrom<this.populationDim){
            putExecutionMode(iChrom);
            infostore.put(ExecutionMode);
            costduration=infostore.GetCostDuration();
            icostduration[iChrom][1]=costduration[1];//duration
            icostduration[iChrom][0]=costduration[0];//additional cost+direct cost
            startArray.add((Vector)infostore.getStartArray());
            iChrom++;
        }
        if(iGen<this.maxGenerations-1)
        {
            evolve();
            startArray.clear();
        }
        iGen++;
    }
    finalResults();
}

public int getPopolationSize(){
    return this.populationDim;
}

public double getFinalCost(int index){
    return icostduration[index][0];
}

public double getFinalDuration(int index){
    return icostduration[index][1];
}

public double getMaxDuration(){
    double maxDuration=icostduration[0][1];//current max
    for(int i=1;i<this.populationDim-1;i++){
        if(icostduration[i][1]>maxDuration){
            maxDuration=icostduration[i][1];
        }
    }
    System.out.println("maxD"+maxDuration);
    return maxDuration;
}

public double getMinDuration(){
    double minDuration=icostduration[0][1];//current min
    for(int i=1;i<this.populationDim-1;i++){

```

Appendix-Java Source Codes for GA Package

```

        if(icostduration[i][1]<minDuration){
            minDuration=icostduration[i][1];
        }
    }
    return minDuration;
}

public double getMaxCost(){
    double maxCost=icostduration[0][0]; //current max
    for(int i=1;i<this.populationDim-1;i++){
        if(icostduration[i][0]>maxCost){
            maxCost=icostduration[i][0];
        }
    }
    return maxCost;
}

public double getMinCost(){
    double minCost=icostduration[0][0];
    for(int i=1;i<this.populationDim-1;i++){
        if(icostduration[i][0]<minCost){
            minCost=icostduration[i][0];
        }
    }
    System.out.println("minC"+minCost);
    return minCost;
}

public Vector getSchedule(int duration){
    Vector schedule=new Vector();
    for(int i=0;i<this.bestFitnessChromIndexes.length;i++){
        if(icostduration[i][1]==(double)duration){
            schedule=(Vector)startArray.elementAt(i);
            break;
        }
    }
    return schedule;
}

protected void calFitness()
{
    for(int i=0;i<this.populationDim;i++){
        System.out.println(icostduration[i][0]); // print cost
    }
    System.out.println();

    for(int i=0;i<this.populationDim;i++){
        System.out.println(icostduration[i][1]); //print duration
    }

    Enumeration e = startArray.elements(); //print startDate for each combin of
                                           //duration&cost
    while(e.hasMoreElements()){
        System.out.println(e.nextElement().toString());
    }
    for(int i=0;i<this.populationDim;i++)

```

Appendix-Java Source Codes for GA Package

```

{
    fitnessArray[i]=0;
    dominatenum[i]=0;
    nondominated[i]=0;
}

for(int i=0;i<this.populationDim;i++)
{
    for(int j=i+1;j<this.populationDim;j++)
    {
        if((icostduration[i][0]>icostduration[j][0]&&icostduration[i][1]>=icostduration[j][1])||
            (icostduration[i][0]>=icostduration[j][0]&&icostduration[i][1]>icostduration[j][1]))
            {
                //was dominated
                dominatenum[j]+=1;
                nondominated[i]+=1;
            }
        else if((icostduration[i][0]<icostduration[j][0]&&icostduration[i][1]<=icostduration[j][1])||
            (icostduration[i][0]<=icostduration[j][0]&&icostduration[i][1]<icostduration[j][1]))
            {
                //dominate other
                dominatenum[i]+=1;
                nondominated[j]+=1;
            }
    }
}

for(int i=0;i<this.populationDim;i++){//calucate the fitness of nondominated,0<=F(i)<1
    if(nondominated[i]==0){
        fitnessArray[i]=dominatenum[i]/(this.populationDim+1);
        bestchroms.addElement(new Integer(i));
    }
}

for(int i=0;i<this.bestchroms.size();i++){//a vector for best chroms exclusively
    int iindex=((Integer)this.bestchroms.elementAt(i)).intValue();
    for(int j=i;j<this.bestchroms.size();j++){
        int jindex=((Integer)this.bestchroms.elementAt(j)).intValue();

        if(icostduration[iindex][0]==icostduration[jindex][0]&&icostduration[iindex][1]==icostduration[jindex][1])
        {
            this.bestchroms.remove(j);
        }
    }
}

for(int i=0;i<this.populationDim;i++){//calculate the fitness of dominated F(i)>1
    double fit=0;
    if(nondominated[i]==0)continue;
    for(int j=0;j<this.populationDim;j++){//find which node dominates it
        if((icostduration[i][0]>icostduration[j][0]&&icostduration[i][1]>=icostduration[j][1])||
            (icostduration[i][0]>=icostduration[j][0]&&icostduration[i][1]>icostduration[j][1]))
            {
                //was dominated
                fit+=fitnessArray[j];
            }
    }
    fitnessArray[i]=fit+1;
}

```

Appendix-Java Source Codes for GA Package

```

    }
    double temp[]=new double[this.populationDim];
    for(int i=0;i<this.populationDim;i++){
        temp[i]=fitnessArray[i];
    }
    Arrays.sort(temp);
    this.bestfitness=temp[0];
    this.worsefitness=temp[this.populationDim-1];

}
protected void putExecutionMode(int iChromIndex)
{
    String sChromosome;

    int position=0;
    int position2=0;

    sChromosome=this.getChromosome(iChromIndex).getGenesAsStr();

    for(int i=0; i<activityNum;i++)
    {
        if(FinishedActivity.contains(Integer.toString(i+1))){delete the finished activities
            ExecutionMode[position2]=0;
            position2++;
            continue;
        }
        for(int j=position;j<(position+activityChoice[i]);j++){
            if(sChromosome.charAt(j)=='1'){
                ExecutionMode[position2]=j+1-position;
                position2++;
                position+=activityChoice[i];
                break;
            }
        }
    }
}

void computeFitnessRankings()
{
    for (int i=0; i < populationDim; i++){
        this.chromosomes[i].fitness = getFitness(i);
    }

    int numBestFitnessChrom=this.bestchroms.size();
    int numWorstFitnessChrom=0;
    for(int i=0; i<populationDim;i++){find the number of best and worst chrom
    {

        if(this.chromosomes[i].fitness==this.worsefitness){
            ++numWorstFitnessChrom;
        }
    }

    this.bestFitnessChromIndexes=new int[numBestFitnessChrom];
    this.worstFitnessChromIndexes=new int[numWorstFitnessChrom];
    int index1=0;

```


Appendix-Java Source Codes for GA Package

```

int index2=0;

for(int i=0; i<populationDim;i++)
{
    if(this.chromosomes[i].fitness==this.worsefitness){
        this.worstFitnessChromIndexes[index2]=i;
        index2++;
    }
}
for(int i=0;i<this.bestchroms.size();i++){
    this.bestFitnessChromIndexes[i]=((Integer)this.bestchroms.elementAt(i)).intValue();
}
}
}

```

TopologicalSort.java

```
package ga;
```

```
import java.util.*;
import java.io.*;
```

```
class Node {
```

```
    String id;
```

```
    Node (String id) {
        this.id = id;
    }
}
```

```
class Arc {
```

```
    Node from;
    Node to;
```

```
    Arc (Node from, Node to) {
        this.from = from;
        this.to = to;
    }
}
```

```
class Digraph {
```

```
    Vector nodes = new Vector();
    Vector arcs = new Vector();
```

```
    boolean isEmpty () {
        return nodes.isEmpty();
    }
}
```

```
    private Node getNode (String id) {
        for (int k = 0; k < nodes.size(); k++) {
            Node node = (Node) nodes.elementAt(k);
            if (node.id.equals(id)) return node;
        }
    }
}
```

Appendix-Java Source Codes for GA Package

```

    }
    return null;
}

Node addNode (String id) {
    Node node = getNode(id);
    if (node == null) {
        node = new Node (id);
        nodes.addElement (node);
    }
    return node;
}

Arc addArc (String fromId, String told) {
    Node fromNode = addNode (fromId);
    Node toNode = addNode (told);
    Arc arc = new Arc (fromNode, toNode);
    arcs.addElement (arc);
    return arc;
}

Vector getpredecessors(String nodeId){
    Vector predecessor=new Vector();
    for(int i=0;i<arcs.size();i++){
        Arc arc=(Arc)arcs.elementAt(i);
        if((arc.to.id).equals(nodeId)){
            predecessor.addElement(arc.from.id);
        }
    }
    return predecessor;
}

void removeArc (Arc arc) {
    arcs.removeElement (arc);
}

void removeNode (Node node) {
    for (int a = arcs.size() - 1; a >= 0; a--) {
        Arc arc = (Arc) arcs.elementAt(a);
        if (arc.from == node || arc.to == node) removeArc (arc);
    }
    nodes.removeElement (node);
}

// isMinimal (node) return true if there is no arc such that node
// is on its 'to' side.
boolean isMinimal (Node node) {
    for (int a = 0; a < arcs.size(); a++) {
        Arc arc = (Arc) arcs.elementAt(a);
        if (arc.to == node) return false;
    }
    return true;
}

// getMinimalNode: return a minimal node or null if no such node exists.
Vector getMinimalNode () {

```

Appendix-Java Source Codes for GA Package

```

        Vector vector=new Vector();
        for (int n = 0; n < nodes.size() ; n++) {
            Node node = (Node) nodes.elementAt(n);

            if (isMinimal (node)) {
                vector.addElement(node);}
        }

        return vector;

    }
}

// -----

public class TopologicalSort {

    protected Vector FinishedActivity;
    protected Vector OnGoingActivityIdMode;
    protected int[] ExecutionMode;
    protected int[] pArray;
    protected int[] OverlapDay;
    protected ActivityInitialization init;
    protected int activityNum;
    protected int modesum;
    protected double duration;
    protected double totalcost=0;
    protected Vector startArray;// constains id,startdate, mode
    protected int numLabor;

    public TopologicalSort(ActivityInitialization init,int[] parray, int[] exemode){
        this.init=init;
        this.pArray=parray;
        this.ExecutionMode=exemode;
        this.FinishedActivity=init.GetFinishedActivities();
        this.OnGoingActivityIdMode=init.GetOngoingActivities();
        this.activityNum=init.GetActivityNum();
        this.modesum=init.GetModeSum();
        this.OverlapDay=init.GetOverlapDay();
        this.numLabor=init.numLabor;
        this.startArray=new Vector();
        for(int i=0;i<exemode.length;i++){
            System.out.print(exemode[i]);
        }
        System.out.println();
    }
    public double GetTotalCost(){
        return (double)totalcost;
    }
    public double GetDuration(){
        return (double)duration;
    }
    public void sort(){

```

Appendix-Java Source Codes for GA Package

```

Hashtable idpriority=new Hashtable();
Hashtable idpredecessor=new Hashtable();
Digraph digraph = new Digraph();
Vector sortedNodes = new Vector();

// load the precedence relationship*****
for(int i=0;i<activityNum;i++){
    for(int j=0;j<modesum;j++){

if((init.GetActivityInfo(j).GetID().equals(init.GetActivitySet(i)))&&init.GetActivityInfo(j).GetExecution
Mode()==1){
    for(int k=0;k<init.GetActivityInfo(j).getNumOfSuccessors();k++){
        String fromId=init.GetActivitySet(i);
        String toId=init.GetActivityInfo(j).GetSuccessors(k);
        digraph.addArc(fromId, toId);
        //System.out.println("prec"+fromId+" succ "+toId);
    }
}

}

}

//remove those finished activity
for(int i=0;i<FinishedActivity.size();i++){
    if(FinishedActivity.elementAt(i)==null)break;
    for(int j=0;j<digraph.nodes.size();j++){
        Node node=(Node)digraph.nodes.elementAt(j);
        if(node.id==FinishedActivity.elementAt(i).toString())
            digraph.removeNode(node);
    }
}
//assign priority and predecessors to activity
int position=0;
for(int i=0;i<activityNum;i++){
    String activityid=Integer.toString(i+1);
    if(FinishedActivity.contains(activityid))continue;
    Vector idpre=digraph.getpredecessors(activityid);
    idpredecessor.put(activityid,idpre);
    idpriority.put(activityid,new Integer(pArray[position]));
    position++;
}

// now do the topological sort
while (! digraph.isEmpty()) {
    // search for a minimal node
    Vector eligible = digraph.getMinimalNode ();

    // remove minimal node from digraph, store it in sortedNodes Vector
    int array[]=new int[eligible.capacity()];

    int temp[]=new int[pArray.length];
    for (int i=0;i<pArray.length;i++){temp[i]=pArray[i];}

    Arrays.sort(temp);

```


Appendix-Java Source Codes for GA Package

```

for(int i=0; i<eligible.size();i++){
    Node key2=(Node)eligible.elementAt(i);
    String id=key2.id;
    int value=((Integer)idpriority.get(id)).intValue();

    array[i]=value;
}
//find the max value in the array
int location=0;
outer:for(int j=((pArray.length)-1);j>=0;j--){

    inter: for(int i=0;i<eligible.capacity();i++){
        if(array[i]==temp[j]){
            location=i;
            break outer;}
    }
}
Node mini=(Node)eligible.elementAt(location);//error may occur
sortedNodes.addElement (mini);
digraph.removeNode (mini);
}

//---calculate the resource-constrained activity duration-----
int resourcestore[]=new int[600];
for(int a=0;a<600;a++){resourcestore[a]=numLabor;}
int start=0;
int finish;
int finisharray[]=new int[sortedNodes.size()];

for(int i=0;i<sortedNodes.size();i++){
    int durationvalue=0;//initialization
    int resourcevalue=0;
    int directcost=0;
    int addcost=0;

    Node node=(Node)sortedNodes.elementAt(i);
    int exemd=Integer.parseInt(node.id)-1;

    for(int c=0;c<modesum;c++){

if((init.GetActivityInfo(c).GetID().equals(node.id))&&(init.GetActivityInfo(c).GetExecutionMode()==
ExecutionMode[exemd])){
    durationvalue=init.GetActivityInfo(c).GetDuration();
    resourcevalue=init.GetActivityInfo(c).ResourceConsumption;
    directcost=init.GetActivityInfo(c).GetCost();
    addcost=init.GetActivityInfo(c).GetAddCost();
    }
    }
    Vector predecessorvalue=(Vector)idpredecessor.get(node.id);

if(i==0) {//no interrupted activity here, as it is the 1st activity to schedule

    finisharray[0]=durationvalue;
    startArray.add(node.id); //id followed by the start date for that activity

```

Appendix-Java Source Codes for GA Package

```

        startArray.add(new Integer(1)); //first activity should start on time
        int modelId=ExecutionMode[Integer.parseInt(node.id)-1]; //the corresponding mode for the id
        startArray.add(new Integer(modelId));
        totalcost+=directcost;
        for(int b=1;b<=durationvalue;b++){
            resourcestore[b]=resourcestore[b]-resourcevalue;
        }

        } //first activity
    else{
        int max[]=new int[sortedNodes.size()];
        for(int j=0;j<predecessorvalue.size();j++){
            String nodeId=(String)predecessorvalue.elementAt(j);
            for(int k=0;k<sortedNodes.size();k++){
                Node node3=(Node)sortedNodes.elementAt(k);
                String nodeId2=node3.id;
                if(nodeId2==nodeId)max[k]=finisharray[k];
            }
        }

        Arrays.sort(max);
        int overlap=0;
        for(int k=0;k<sortedNodes.size();k++){ //for overlap
            Node nodee=(Node)sortedNodes.elementAt(k);
            String nodeeid=nodee.id;
            if(predecessorvalue.contains(nodeeid)&&(max[(sortedNodes.size()-1)]==finisharray[k])){
                overlap=OverlapDay[Integer.parseInt(nodeeid)-1];
                //here only 1 precedence is allowed to be overlapped, pay attention to the activity setting
            }
        }

        int lateststart=max[sortedNodes.size()-1]+1-overlap;
        int planstart=lateststart;
        for(int l=0;l<durationvalue;l++){
            if(resourcestore[lateststart+l]<resourcevalue){
                lateststart=lateststart+l+1;
                l=-1;
            }
        }
        if(planstart!=lateststart){
            if(OnGoingActivityIdMode.contains(node.id)){totalcost+=directcost+addcost;}
            else{
                totalcost+=directcost;
            }
        }else{
            totalcost+=directcost;
        }
        finisharray[i]=lateststart+durationvalue-1;
        startArray.add(node.id); //id followed by the start date for that activity
        startArray.add(new Integer(lateststart));
        int modelId=ExecutionMode[Integer.parseInt(node.id)-1]; //the corresponding mode for the id
        startArray.add(new Integer(modelId));

        for(int c=lateststart;c<(durationvalue+lateststart);c++){

```

Appendix-Java Source Codes for GA Package

```

        resourcestore[c]=resourcestore[c]-resourcevalue;
    }
}

duration=finisharray[(sortedNodes.size()-1)];
}
public Vector getStartDate(){return startArray;}
}

```

GeneticAlgorithms.java

```

package ga;
import ga.*;
import plan.*;

import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.util.ArrayList;
import java.util.Vector;
import java.lang.Integer;
import javax.swing.tree.*;

import javax.swing.border.EtchedBorder;
import javax.swing.border.TitledBorder;

public class GeneticAlgorithms extends JDialog
{
    //general panel
    private JPanel generalPanel;

    private GaSetting set;

    //first row
    private JPanel firstRowPanel;
    private JPanel timecostPanel;
    private JLabel populationLabela;
    private JLabel generationLabela;
    private JLabel crossoverLabela;
    private JLabel mutationLabela;
    private JTextField tcPop;
    private JTextField tcGen;
    private JTextField tcCro;
    private JTextField tcMut;

    //second row
    private JPanel secondRowPanel;
    private JPanel resourcePanel;
    private JLabel populationLabelb;
    private JLabel generationLabelb;

```

Appendix-Java Source Codes for GA Package

```

private JLabel crossoverLabelb;
private JLabel mutationLabelb;
private JTextField rsPop;
private JTextField rsGen;
private JTextField rsCro;
private JTextField rsMut;

//third row for start and cancel button
private JPanel thirdRowPanel;
private JButton startButton;
private JButton cancelButton;
private JButton displayButton;
private JTextField jtfDuration;
private Vector schedule;
private TimeCostTradeOff opt;
private ActivityInitialization init;

//fourth row for displaying the time-cost plot
private JPanel fourthRowPanel;
private Image imgCostDuration = null;

//fifth row for progress bar
private JPanel fifthRowPanel;

//last row for selecting duration
private JPanel lastRowPanel;
private JLabel durationLabel;
private JTextField duration;
private JButton okButton;

GridBagConstraints gbc = new GridBagConstraints();
FlowLayout flowL = new FlowLayout(FlowLayout.LEFT, 10, 10);

/** ArrayList to hold all the tasks */
private ArrayList all;
/** gantt chart area of the application */
private GanttChartPanel aarea;
/** GanttTree of the application. */
private GanttTree tree;
/** Crew of the application */
private GanttPeoplePanel ppeop;
JFrame prj;
/**true if the ok button was pressed */
public boolean change = false;

/**add a component to container by using GridBagConstraints.*/
private void addUsingGBL(Container container, Component component, GridBagConstraints gbc,
int x, int y, int w, int h) {
    gbc.gridx = x;
    gbc.gridy = y;
    gbc.gridwidth = w;
    gbc.gridheight = h;
    gbc.weighty = 0;
    container.add(component, gbc);
}

```


Appendix-Java Source Codes for GA Package

```

/** Construtor */
public GeneticAlgorithms(JFrame parent,GanttTree tree,GanttPeoplePanel
peop,GanttGraphicArea area,GanttLanguage language)
{
    super(parent,"Genetic Algorithms",true);
    this.ttree = tree;
    this.aarea = area;
    this.lang = language;
    this.prj=parent;
    this.ppeop=peop;

    generalPanel = new JPanel(new GridBagLayout());

    //first row
    set=new GaSetting();
    populationLabela=new JLabel("Population:");
    generationLabela=new JLabel("Generation:");
    crossoverLabela=new JLabel("Crossover Rate:");
    mutationLabela=new JLabel("Mutation Rate:");
    tcPop=new JTextField(5);
    tcGen=new JTextField(5);
    tcCro=new JTextField(5);
    tcMut=new JTextField(5);
    timecostPanel = new JPanel(flowL);
    timecostPanel.add(populationLabela);
    timecostPanel.add(tcPop);
    timecostPanel.add(generationLabela);
    timecostPanel.add(tcGen);
    timecostPanel.add(crossoverLabela);
    timecostPanel.add(tcCro);
    timecostPanel.add(mutationLabela);
    timecostPanel.add(tcMut);
    firstRowPanel=new JPanel();
    firstRowPanel.setBorder(new TitledBorder(new EtchedBorder(),
        "time-cost trade-off subsystem"));
    firstRowPanel.add(timecostPanel);

    //second row
    populationLabelb=new JLabel("Population:");
    generationLabelb=new JLabel("Generation:");
    crossoverLabelb=new JLabel("Crossover Rate:");
    mutationLabelb=new JLabel("Mutation Rate:");
    rsPop=new JTextField(5);
    rsGen=new JTextField(5);
    rsCro=new JTextField(5);
    rsMut=new JTextField(5);
    resourcePanel=new JPanel(flowL);
    resourcePanel.add(populationLabelb);
    resourcePanel.add(rsPop);
    resourcePanel.add(generationLabelb);
    resourcePanel.add(rsGen);
    resourcePanel.add(crossoverLabelb);
    resourcePanel.add(rsCro);
    resourcePanel.add(mutationLabelb);
    resourcePanel.add(rsMut);
    secondRowPanel=new JPanel();

```

Appendix-Java Source Codes for GA Package

```

secondRowPanel.setBorder(new TitledBorder(new EtchedBorder(),
    "resource scheduling subsystem");
secondRowPanel.add(resourcePanel);

//third row
thirdRowPanel=new JPanel(flowL);
schedule=new Vector();
startButton=new JButton("Start", new ImageIcon(getClass().getResource("yes.png")));
getRootPane().setDefaultButton(startButton);
thirdRowPanel.add(startButton);
cancelButton=new JButton("Cancel", new ImageIcon(getClass().getResource("no.png")));
thirdRowPanel.add(cancelButton);
thirdRowPanel.add(new JLabel(" The schedule with"));
jtfDuration= new JTextField(4);
thirdRowPanel.add(jtfDuration);
thirdRowPanel.add(new JLabel("days"));
displayButton=new JButton("Display", new ImageIcon(getClass().getResource("yes.png")));
displayButton.setEnabled(false);
thirdRowPanel.add(displayButton);

//fourth row
fourthRowPanel=new JPanel(flowL);

//Listener on start button
startButton.addActionListener(new ActionListener()
{public void actionPerformed(ActionEvent evt)
{
    set.setTcPop(new Integer(tcPop.getText()).hashCode());
    set.setTcGen(new Integer(tcGen.getText()).hashCode());
    set.setTcCross(new Double(tcCro.getText()).doubleValue());
    set.setTcMutation(new Double(tcMut.getText()).doubleValue());

    set.setRsPop(new Integer(rsPop.getText()).hashCode());
    set.setRsGen(new Integer(rsGen.getText()).hashCode());
    set.setRsCross(new Double(rsCro.getText()).doubleValue());
    set.setRsMutation(new Double(rsMut.getText()).doubleValue());

    imgCostDuration = null;

    paint(getContentPane().getGraphics());

    System.out.println("Starting Cost Duration Optimization...");
    try
    {
        InformationStore is=new InformationStore();
        init=new ActivityInitialization(ttree,ppeop);
        opt=new TimeCostTradeOff(set,is,init);
        ResourceScheduling duration=new
ResourceScheduling(set,is,init,init.geneValue,set.getTcPop(),set.getTcGen());//population....gene
ration...
        Thread optthread=new Thread(opt);
        Thread durationthread=new Thread(duration);
        optthread.start();
        durationthread.start();
        try{optthread.join();}catch(InterruptedException ie){}
        plotTimeCostCurve(opt);
    }
}
}

```

Appendix-Java Source Codes for GA Package

```

    }
    catch (GAEException gae)
    {
        System.out.println(gae.getMessage());
    }

    }

});

cancelButton.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent evt)
    {
        setVisible (false); }
});

displayButton.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent evt)
    {
        int iduration=Integer.parseInt(jtfDuration.getText());
        schedule=opt.getSchedule(iduration);
        System.out.println(schedule);
        for(int i=0;i<schedule.size();i+=3){//id,startdate,mode
            int taskId=Integer.parseInt((String)schedule.elementAt(i));
            GanttTask ttask=ttree.getTaskByNumber(taskId);
            int startDate=((Integer)schedule.elementAt(i+1)).intValue();
            ttask.setStart(new GanttCalendar(startDate));
            int mode=((Integer)schedule.elementAt(i+2)).intValue();
            ttask.setSelectedRow(mode-1);
            ttask.setExecutionmode(mode);
            String[][] data=new String[6][5];
            data=ttask.getData();
            ttask.setLength(Integer.parseInt(data[mode-1][1]));
            ttask.setLabor(Integer.parseInt(data[mode-1][2]));
            ttask.setEngineer(Integer.parseInt(data[mode-1][3]));
            ttask.setOverlap(Integer.parseInt(data[mode-1][4]));

        }
        aarea.repaint();
    }
});

gbc.anchor = GridBagConstraints.WEST;
gbc.insets.right = 15;
gbc.insets.left = 10;
gbc.insets.top = 10;
gbc.weighty = 0;
addUsingGBL(generalPanel, firstRowPanel, gbc, 0, 0, 1, 1);
addUsingGBL(generalPanel, secondRowPanel, gbc, 0, 1, 1, 1);
addUsingGBL(generalPanel, thirdRowPanel, gbc, 0, 2, 1, 1);

getContentPane().add(generalPanel, BorderLayout.NORTH);
getContentPane().add(fourthRowPanel, BorderLayout.SOUTH);

```


Appendix-Java Source Codes for GA Package

```

pack();
show();
setResizable(false);
Point point = parent.getLocationOnScreen();
int x = (int)point.getX() + parent.getWidth()/2;
int y = (int)point.getY() + parent.getHeight()/2;
setLocation(x - getWidth()/2, y-getHeight()/2);
}

void plotTimeCostCurve(CostDurationOpt opt)
{
    int xPos, yPos;
    double xScaleFactor, yScaleFactor;
    int xDim = 350;
    int yDim = 200;

    imgCostDuration = createImage(xDim, yDim);
    imgCostDuration.getGraphics().drawRect(35, 0, xDim-40, yDim-13); //x,y,width,height

    double maxDuration=opt.getMaxDuration();
    double minDuration=opt.getMinDuration();
    double maxCost=opt.getMaxCost();
    double minCost=opt.getMinCost();

    yScaleFactor = (double)(yDim-24) / (maxCost-minCost);
    xScaleFactor = (double)(xDim-35) / (maxDuration-minDuration);

    imgCostDuration.getGraphics().drawString(""+(int)minDuration, 35, yDim); //string,x,y
    imgCostDuration.getGraphics().drawString(""+(int)maxDuration, xDim-25, yDim);
    imgCostDuration.getGraphics().drawString("Duration", xDim / 2 - 15, yDim);
    imgCostDuration.getGraphics().drawString(""+(int)minCost,0,yDim-11);
    imgCostDuration.getGraphics().drawString(""+(int)maxCost,0,8);
    imgCostDuration.getGraphics().drawString("Cost", 12,yDim/2);

    for (int i=0; i < opt.getPopolationSize(); i++)
    {
        xPos = 35+(int)(xScaleFactor * (opt.getFinalDuration(i)-minDuration));
        yPos = yDim - ((int)(yScaleFactor * (opt.getFinalCost(i)-minCost))+18);
        imgCostDuration.getGraphics().setColor(Color.red);
        imgCostDuration.getGraphics().drawOval(xPos, yPos, 5, 5);
    }

    repaint(); //calls paint() and displays plot images
    displayButton.setEnabled(true);
}

public void paint(Graphics g)
{
    super.paint(g);
    if (imgCostDuration != null)
        g.drawImage(imgCostDuration, 10, 275, this); //img,x,y,observer
}
}

```