Capability-based software project scheduling with system dynamics and heuristic search

by

Yujia Ge

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Iowa State University

Ames, Iowa

2004

Graduate College
Iowa State University

This is to certify that the master's thesis of

Yujia Ge

has met the thesis requirements of Iowa State University

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Software project management, a peculiar area in project management, is concerned with activities involved in ensuring that software is delivered on time and on schedule. Software project management is unique because software products are perceived as intangible and flexible. In software projects the capability of people is the most important factor to determine whether a project will succeed or fail. During the software project management process, scheduling can be influenced by a lot of dynamics elements such as the skills of engineers, the growth of skills and experiences, cooperation and leadership. Our objective is to help project managers to assign human resources automatically and realistically based on personnel/team capability.

This thesis proposes a framework for scheduling and monitoring in software project management. Based on this framework, dynamic elements in software project management, especially personnel capability, are simulated in System Dynamics models. The genetic algorithms for previous work (i.e., task-based model and timeline-based model) are revised to reduce the computation in the new model. Experiments are also reported including the procedure on tuning GA parameters, results of several example tests, and discussion on the results. Experiments on comparison of GA and Hill-Climbing method help us build the confidence that GA is a good choice in software project scheduling problems.

CHAPTER 1. Introduction

1.1 Objective

Software project management, a peculiar area in project management, is concerned with activities involved in ensuring that software is delivered on time and on schedule. Software project management is unique because software products are perceived as intangible and flexible. Thus, software engineering is neither a classical engineering discipline, as is mechanical or electrical engineering, nor does it dictate a standardized software development process. These characteristics make software project management very difficult to satisfy budget and schedule constraints that are set by an organization and its stakeholders. As reported by Standish Group, only about 28% of software projects in US companies succeeded in 2000 and more than 40% were canceled before completion [43].

In software projects, the capability of people is the most important factor to determine whether a project will succeed or fail. During the software project management process, scheduling can be influenced by a lot of dynamic elements such as the skills of engineers, the growth of skills and experiences, cooperation and leadership. Current resource-constrained scheduling techniques mostly focus on the availability of resources instead of the capability of resources. It is obviously not reasonable and incomplete, especially in software project management where the capability of human resources can evolve during the execution of an established plan. Ignoring learning and other dynamic factors during project estimation can lead to wrong execution. Our research aims to define and explore the software project scheduling by considering the dynamics throughout the software development life cycle.

Our objective is to help project managers assign human resources automatically and realistically based on personnel/team capability. We propose a framework for capability-based software project scheduling to model software processes at the micro levels using the system dynamics method. Given a certain project and available resources, heuristic search is applied as an optimization method to obtain a near-optimal solution under the framework. In our future work, we also try to explain how case-based methods can provide feedback to the remaining tasks during the execution of a project for more accurate simulation.

1.2 Scope of Our Research

Project management includes those activities of scheduling, planning and monitoring resources (including human resources) to achieve specific objectives. The usual goal is to obtain a schedule which can maximize the objective, provided that there is a way to evaluate performance. The difference between a plan and a schedule can be considered as follows. A plan defines *what* must be done and restrictions on *how* to do it (estimates), while a schedule specifically describes both *how* and *when* will the task be done (temporal assignments) [48]. Although planning and scheduling are traditionally considered as independent activities, we cannot separate them in reality. Our research scope is mostly on scheduling, but at the same time we have to refer to planning as well. Figure 1.1 shows the range of software project management processes and where our research is focused.

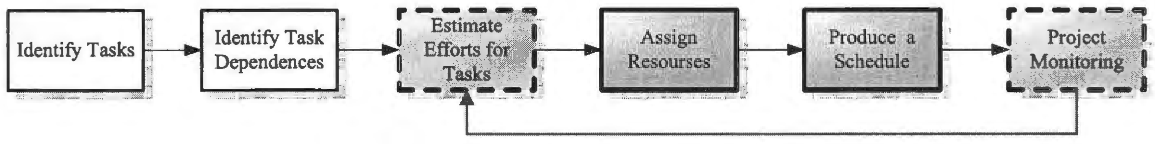

Figure 1.1 Project management process

Although scheduling is tedious and error-prone for software managers, software project management receives relatively little attention in the software engineering research community. Therefore our research emphasizes the scheduling problems of software development. Because the general resource-constrained project scheduling problem lacks the characteristic of the software development environment, we propose a capability-based scheduling model and apply optimization techniques based on this model as depicted by the boxes "assign resources" and "produce a schedule" in Figure 1.1. The other two shaded boxes "estimate efforts for tasks" and "project monitoring" will be partially related to our model. Due to the lack of real software project data, we use simulation to validate our algorithms.

1.3 Structure of This Thesis

The thesis is organized as follows: Chapter 2 gives a literature review over the related research on project management. Chapter 3 describes limitations of previous work in our group, i.e. the task-based model and timeline-based model. Chapter 4 briefly illustrates the framework of our method. Chapter 5 describes our model in detail, including the dynamic model and the static model. Chapter 6 focuses on the algorithm and implementation of heuristic search and discusses experimental results. Chapter 7 gives an overall evaluation on our current research and concludes with some future directions.

CHAPTER 2. Related Work

2.1 Resource-Constrained Project Scheduling Problems

In many industrial applications such as manufacturing, production planning, project management and elsewhere, project scheduling problems occur with limited resource availability. For a general resource-constrained project scheduling problem, it is described as a project with a set of tasks, or activities. Tasks have precedence relationships. Tasks also have estimated durations and may include various other measures such as cost. We need to assign the tasks to some resources to meet our predefined objectives mostly as minimal *makespan* such that both the precedence and resource constraints are fulfilled. The makespan is the time needed to complete all the tasks. An example of a task precedence graph is shown in Figure 2.1. The project consists of 6 tasks with constraints including: the required skills to finish Task 1 are Java and Microsoft Project; Tom's salary is higher than Mike's; Jenny works on Monday, Tuesday, and Friday.

Resource-constrained project scheduling problem, or RCPSP, has been thoroughly studied for more than 40 years. An overview of the different models in RCPSP is given by Brucker *et al* [9]. Young *et al* [50] summarizes the different types of RCPSP problems, research directions, algorithm and heuristic approaches presented in recent years. Young also describes a new problem definition with the objective of minimum cost.

2.1.1 Exact Solution Methods

When resource-constrained scheduling solutions were first proposed, simple models were used with exact methods for solving problems. Exact methods try to find optimal solutions through some intelligent exhaustive search. They include backtracking [6], branch and bound [8] or

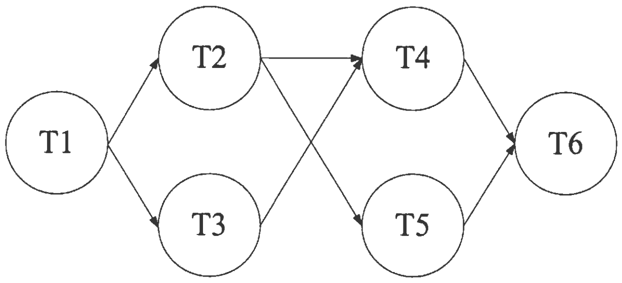

Figure 2.1 An example of task precedence graph

implicit enumeration, the critical path method and its variations, and dynamic programming [4]. Given a problem, the exact methods can find the best solution if it exists. However, when constraints are added, the difficulty of solving a problem increases. In addition, significant problem size greatly affects the feasibility of those methods. For example, the critical path method was devised for finding the shortest time to complete a project given estimates of task durations. Unfortunately, the critical path method cannot solve problems that include restrictions on the number of resources that are available. Recent research on exact solutions includes a tree search algorithm reported by Mingozzi [34].

2.1.2 Heuristic Solution Methods

Although heuristic methods may not find optimal solutions compared to exact methods, they can still find good solutions with less time. In general, heuristic methods require more space. Heuristics are rules to help make a decision given a particular situation. Heuristics in scheduling are usually referred to as scheduling rules or dispatch rules. Heuristics could be deterministic or stochastic. There are several common heuristic approaches to scheduling problems. Simulated Annealing (SA), introduced by Kirkpatrick *et al* [29], originated from the physical annealing process. It requires a schedule representation as well as a neighborhood operator for moving from the current solution to a candidate solution. Tabu Search (TS) developed by Glover [22] is essentially a search method for guiding known heuristic to overcome local optimality. Genetic Algorithm (GA), inspired by the process of biological evolution, was introduced by Holland [26]. The advantage of GA is that it can handle arbitrary kinds of constraints and objectives that can all be treated as weighted components of the fitness function. GA has been extensively applied in scheduling problems [32], [50]. In one of the earliest published works on the application of GAs to scheduling, Davis [15] outlines a basic scheme applied to a simplified toy problem in flow-shop scheduling. Later different GA algorithms are applied in variations of the general resource-constrained scheduling problems [48], [37], [24]. We focus on the application of GA to the scheduling problem in software development in this thesis.

2.1.3 Problems Related to Real Software Project Scheduling

Researchers tend to adopt simplified models for problem solving with some exact algorithms. However, in the real-world situations, the problem is unfortunately not simple. As we have stated in our objective, the resource-constrained scheduling problem is not complete for software engineering research. Current research in the general resource-constrained scheduling area still focuses mostly on the optimization methods, such as techniques of using Genetic Algorithms (GAs) without considering the specific situations in software development. The main problem is that the model must be sound and applicable in the software engineering process. The model requires the specification of a set of "essential" project information to support decisions and optimization. It is clear that we cannot model every single element of the entire software engineering process. We will describe our model in detail in later sections.

2.2 System Dynamics on Software Project Management

System Dynamics (SD) is a method to model a system by using feedback loops. Roberts [39] defines SD as the application of feedback control systems principles and techniques to managerial, organizational, and socioeconomic problems. This rigorous modeling technique begins with levels or populations (for example, effort to finish a task), and determinants of rates of flow of populations (for example, actual productivity according to resources). SD is a set of conceptual tools that enable us to understand the structure and dynamics of complex systems [44]. It enables us to build formal computer simulations of complex systems and use them to design more effective policies and organizational systems for the future, by effectively addressing dynamic structures. Some simulation tools such as Vensim, Dynamo, iThink, are popular, but most system dynamics are not widely applied in the industry.

Software process modeling is a subject of mathematical modeling in a less mature process. Feedback mechanism may play an important role in software development. For example, schedule pressures causing an employee to speed up work can continually affect the whole software project development process. Additionally, some other parameters need to be updated and tracked during the process. Therefore, the continuous feedback loops need to be modeled in a realistic project management. Since the first application of system dynamics by Abdel-Hamid [1] on project management, there has been some other recent extension work on system dynamics within the realm of project management, such as the hybrid software process simulation model [30] and system dynamics extension modules [3]. Most of the research objectives are designed so that the researcher may know the software process better. As we have a different goal and view for our specific concerns with project management, we will use the system dynamic model to simulate the dynamic part of project management.

2.3 Cost Estimation Models

For scheduling a software project, several cost estimation approaches are in use, such as Putnam's SLIM model and Boehn's COCOMO, and the revision COCOMO II. Among those traditional static cost estimation models, COCOMO (COnstructive COst MOdel), developed by Boehm in 1981, is considered the most complete and popular model. Those empirical approaches involve models fitted to historical data. The models are then used to predict the cost of future projects. The main equation in COCOMO is,

$$Effort = a * (Size)^b \tag{2.1}$$

where a and b are empirical constants derived from the calibration of the model. It has been formulated as a hierarchy of models as Basic COCOMO, Intermediate COCOMO, and Advanced COCOMO. The latter one was embedded with more cost drivers due to more information learned during later development stages.

In spite of the popularity of COCOMO and COCOMO II, we still can see the limitations of these approaches. One report showed that the predictive accuracy of COCOMO II is only 30 percent of the actual values 52 percent of the time for "effort" [13]. Several possible ways can be used to improve the accuracy of estimation, such as introducing more factors and adapting the existing approaches to new development techniques.

Another approach of estimation is to model the system process using system dynamics which we have described the techniques in Section 2.2. Table 2.1 is a comparison of COCOMO and system dynamics stated by Roman and Carrieira [40].

Table 2.1 Comparison between COCOMO models and dynamic cost estimation model

Static Cost Estimation Models (COCOMO)	*Dynamic Cost Estimation Models*
Basic COCOMO	RDM (Reduced Dynamic Model)
Intermediate COCOMO	Abdel-Hamid and Madnick's, Draper Laboratory, SEPS, etc
Advanced COCOMO	

Although Roman and Carrieira [40] believe there are no dynamics cost estimation models to parallel the power of advanced COCOMO, we would like to use more detailed system dynamics to do similar work at the same level of the advanced COCOMO. Our work is not to replace the COCOMO model, but to try to complement it at a detailed level.

2.4 Project Scheduling Tools

There are many commercial project management tools such as *Microsoft Project*, Symantec Corporation's *Time Line* and the web-based project management tools of Rational Concepts. None of these, however, provides automatic scheduling functionality. The only software to help automatic scheduling for project management that we can find is *Opensched*. It reads a file describing the project as input and produces textural descriptions of the generated project plan, Gantt charts and network diagrams. The input includes tasks which must be accomplished, resources (e.g., people, equipment, and facilities) which may work on tasks and work that has already been completed. However, the model supported in *Opensched* is very simple.

Usually a scheduling tool needs to be consistent with the software development methodology, such as the Rational Unified Process. Although it is not always practical to use automated project scheduling in project management, research is still needed for improving the overall capabilities of current tools.

CHAPTER 3. Previous Work and Limitations

Our previous two models, the task-based model and the timeline-based model have made a serious attempt to model realistic software scheduling processes.

3.1 Task-based Model

The task-based model [10] is proposed as an improvement to the original model (i.e. SPMNet [12]). It uses Genetic Algorithm as its optimization method.

The representation of the problem consists of:

1) Representation of project $TPG = (V, E)$:

- The project is represented as a task precedence graph;
- A directed acyclic graph where the nodes represent the tasks and the edges represent the task precedence;
- Each task is associated with an estimated effort (based on COCOMO) and the required skills.

2) An employee database D_{emp} with information of skills and salary.

3) An objective function.

In this model, genetic representation is an orthogonal 2D array with one dimension for tasks, the other for employees. GA operators are adopted from GAlib [18]. In the approach, there are two stages for scheduling the project. The first stage evaluates how the genome satisfies the constraints while the second stage evaluates the schedule performance of the genome. For the simplest objective function, it can be defined as: *Composite objective function = Validity * (OverLoadWeight/OverLoad+ MoneyWeight/CostMoney + TimeWeight/CostTime).*

Validity (validity of job assignments) is usually scored on a 0/1 basis, 0 if the assignments are invalid, 1 if they are valid. *OverLoad* (minimum level of overtime) is the amount of time worked beyond the individual overtime limits summed over all employees, and it is treated as a global objective for a project. *CostMoney* (minimum cost) is the total labor cost of performing the project computed using the labor rates of each resource and the hours applied to the tasks. *CostTime* (minimum of time span) is the total time span required to finish the project from the start of the first task until the end of the last. The composite objective value is the summation of weighted component objective values.

3.2 Timeline-Based Model

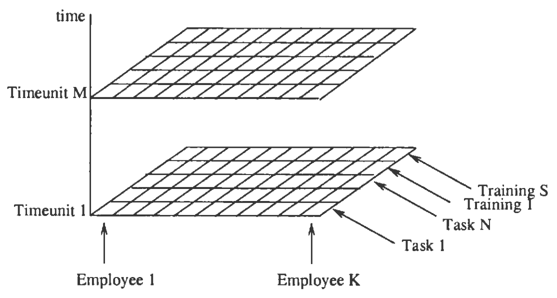

Figure 3.1 Assignment in timeline-based model

In timeline-based model [17], a timeline is introduced to improve the original model in task-based model [10]. The timeline expands the two-dimensional (task and employee) model to a three-dimensional one which shows the effort of each employee applied to each task in each time unit. The timeline helps capture the dynamic nature of software management, such as re-assignment of employees, learning, scheduled vacation, unexpected leave, suspension and resumption of tasks, and the introduction of hard, intermediate deadlines.

Genetic representation for timeline-based model is a 3D array. Figure 3.1 illustrates the scheme for Employee-Task Assignment. The computational complexity of the timeline-based model is sharply increased because of the introduction of time dimension compared to the original task-based model. In order to achieve realism, the elements, such as an employee's proficiency scores, employees' technical experiences, task deadlines, and task penalties are considered. Models related to employees were added, such as *Employee Model* (an employee represented by a numerical identifier and some properties) with *Employee Compensation Model*, *Employee Skill List*, *Employee Training Model*, *Employee Experience Model* and *Availability Model.* Models related to tasks (a task represented as a numerical identifier and some properties) include *Task Estimated Effort*, *Task Importance Model*, *Skill List* and *Ancestor Task List*, and *Maximum Headcount.*

3.3 Limitations

Limitations of the task-based model [10] have already been reported in the timeline-based model [17]. Our own previous work has not adequately dealt with specific characteristics in the software engineering environment. The work is mostly done on general project management scheduling, although our research focus is on software project management.

Here we want to discuss limitations of our later work, the timeline-based model. Generally speaking, the work is unrealistic, mostly because it ignores the effect of the changing staffing profile too often by neglecting the cost to those changes.

Problems of timeline-based scheduling are as follows:

1. Usually a task is assigned to an employee until the task is finished. It is unrealistic to assign an employee to one task for awhile, assign him to another task, and then assign the employee to work on the original task again.

2. The interruption cost of this scheme is not counted, which is very expensive to project execution. Those costs include the interruption on learning and the cost of reassigning an employee.

3. Calculating the fitness function by adjusting gained effort during execution places inaccuracy into the calculation.

4. Re-scheduling is not considered in the framework. The situation is not unusual due to the change of employees and even tasks.

CHAPTER 4. Overview of the Scheduling Framework

This framework introduces a hybrid software process simulation model that combines dynamic and static models in software project scheduling. There are four main parts in this framework as illustrated in Figure 4.1:

(1) Heuristic search algorithms. The key algorithm to implement automatic scheduling is heuristic search. In our work genetic algorithm is used as the major method (refer to Chapter 6). With the information on the properties of tasks and employees from static models (part 3) as the input, heuristic search algorithms can generate populations of possible solutions using genetic operators. Evaluating those individuals by an objective function using the output "durations of tasks" from system dynamics simulation (part 2), they overall evolve to be the ones with better performance after a number of generations. Finally, a near-optimal schedule can be obtained using heuristic search algorithms.

(2) System dynamic models and simulation. Based on system dynamics meta models of team productivity, the capability-based system dynamics simulation is guided to calculate task durations according to different human resources assignment. It provides the task durations to heuristic search algorithm (part 1) as a part of input to calculate the fitness score of an individual (refer to Chapter 5).

(3) Static models. It includes the static part of task models (task estimated effort, task penalty model and required task skill lists, etc.) and employee models, such as the employee payment model, the employee skill model (also has a dynamics characteristic, refer to Chapter 5), and an employee experience model.

(4) Monitoring and re-scheduling techniques. From an actual project execution, some monitoring data can be obtained from daily records. These data can help calibrate system dynamics

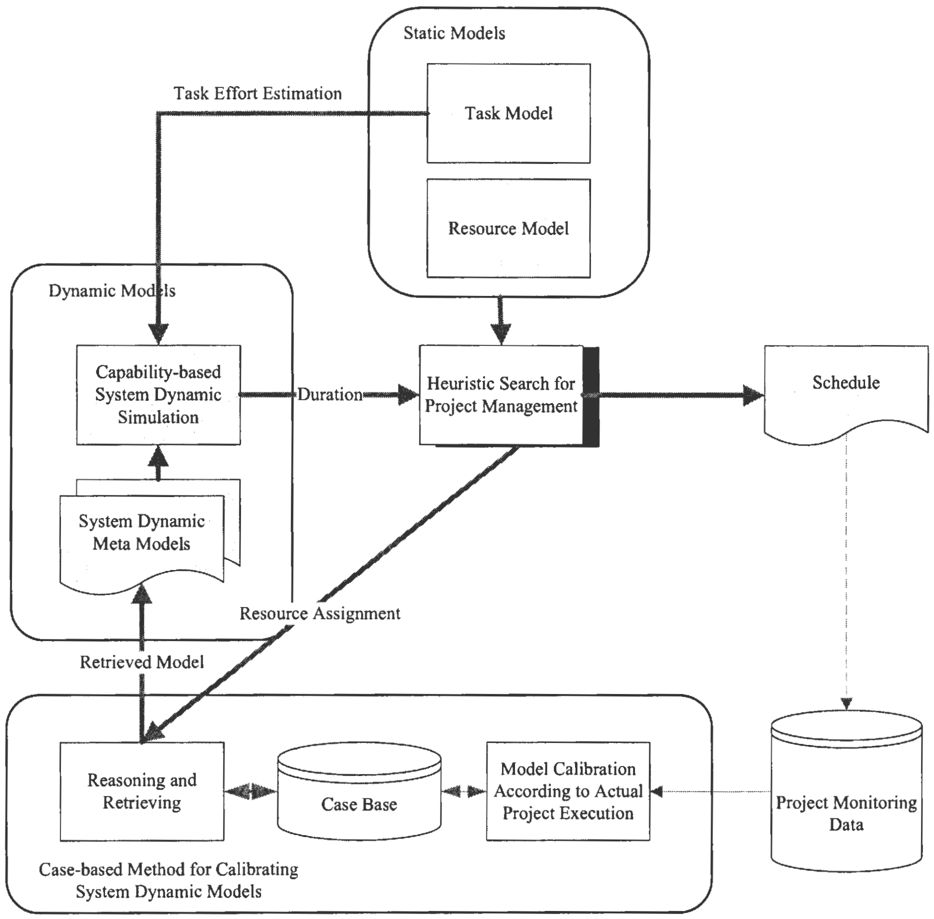
Static Models
Task Effort Estimation
Task Model
Resource Model
Dynamic Models
Capability-based System Dynamic Simulation
Duration
Heuristic Search for Project Management
Schedule
System Dynamic Meta Models
Resource Assignment
Retrieved Model
Reasoning and Retrieving
Case Base
Model Calibration According to Actual Project Execution
Project Monitoring Data
Case-based Method for Calibrating System Dynamic Models

Figure 4.1 Capability-based project scheduling framework

meta models (part 2) with more accurate parameters (refer to Chapter 7). Our assumption is that it is more reasonable and accurate to use data from the same project or the same company. Our rough idea is to use case-based techniques to store historical data as cases which can be retrieved later. As we have not done enough work on model calibration and re-scheduling techniques, it will be paid much more attention in our future work.

Considering the dynamics of team capability, a near-optimal schedule can be generated for decision making in software project management. This framework also gives a possible way to take advantage of actual project execution and use those data as feedback loops for getting more accurate schedules.

CHAPTER 5. Capability-Based Model

The ability of a software organization to take on a new project depends to a large extent on the capability, rather than, the availability of resources [35]. The personnel/team capability is one of the most important factors in software engineering as reported by Software Engineering Center of USC [7]. Our proposed scheduling model is based on human resources' capability with dynamics factors.

5.1 Productivity Factors on Human Resources

As to the productivity metrics, productivity can be measured in the following ways: 1) to use project size (usually measured as lines of code, i.e., LOC), or 2) function points, divided by the time spent on development.

There are significant productivity differences among individual software developers. In the first study on the subject by Sackman [42], they found dramatic differences of more than 20 to 1 in the time required by different developers to debug the same problem. Some researchers have been studying on productivity, but not so many have studied quantitative data. One quantitative study is on object-oriented productivity as reported in Potok [36].

By analysis of the COCOMO model, we can also see that the COCOMO II estimation model includes cost drivers with the capabilities of human resources. Among those factors, 7 out of 22 factors in the COCOMO II model are related to personnel as listed in Table 5.1 [31].

Stevenson [45] gives a literature survey about productivity in software engineering. The factors affecting productivity include organizational structure, office environment and hardware, software tools, people (shortage, turnover, training, experience and innate ability), quality and management. Usually those factors can be divided as task-typed variables and resource-typed

Table 5.1 Personnel influences in the COCOMO II estimation model

COCOMO II Factor	*COCOMO II Name*	*Influence*
Analyst experience	AEXP	1.51
Language and tool experience	LTEX	1.43
Programmer experience	PEXP	1.40
Communications factors	SITE	1.52
Personnel continuity	PCON	1.59
Programmer capability	PCAP	1.77
Requirements analyst capability	ACAP	2.00

variables. As our area is on software management, we focus on factors related to people here. We propose using system dynamics model to represent the factors about an employee's capability.

5.2 Models on System Dynamics

System dynamics models are so complicated because of the introduction of hundreds of interrelated factors. Realizing that it is difficult to model the whole system, a meta model is introduced to help simplify the issue of the creation of complex dynamics models. A meta model is a high-level model for system dynamics. Due to the capability-based characteristic, our meta model for capability will focus on modeling productivity and consists of sub-models at the lower level. We will use a simple and useful parameter that takes value 0 or 1 to control whether we need to consider the factor or not.

5.2.1 Meta Model for Productivity

For software development, team average productivity is the key component affected by a complex set of factors as shown in Figure 5.1.

Our model has three layers: (1) team productivity, which decides the task duration directly; (2) individual productivity, communication overhead, and other factors, which are different among individuals but also contribute to the team productivity; (3) experience, learning, overworking, and schedule pressure factors, which affect the individual productivity on the second level. Figure 5.1 illustrates our meta model with the factors we are currently modeling. There

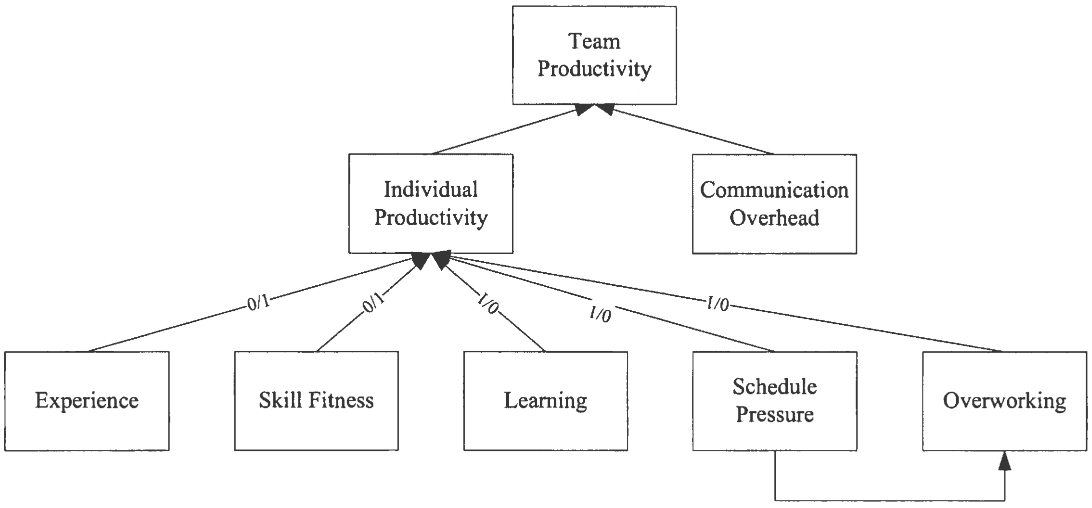

Figure 5.1 Meta model for capability-based model

are also other factors which affect the projects. For example, motivation is also a critical factor to individual productivity. Here, we are not modeling more factors but simplify the models for the purpose of illustrating our framework.

Because project management depends on the ability of managers, the determination of factor influences is partially controlled by project managers through a control parameter. Control parameter takes value 0 or 1 to turn the factors on or off.

5.2.2 Sub Models

While most of the factors would vary from organization to organization and from project to project within a single organization, they would remain constant within a single project [1]. Therefore we introduce the feedback loop to monitor the project as follows.

- Team productivity

$$P_{team} = \sum_{i=1}^{N} P_i * (1 - ComOverhead(N)\%) \tag{5.1}$$

Explanation: Team productivity (P_{team}) describes the relationship between productivity in team and related factors, such as individual productivity (P_i) and communication overhead ($ComOverhead(N)$). Here N is the number of employees assigned to the task. In [1], the productivity of the software development group can be stated as the psychological model of group productivity by Ivan Steiner. In his model:

$$\text{Actual Productivity} = \text{Potential Productivity} - \text{Losses Due to Faulty Process} \tag{5.2}$$

where losses due to faulty process refer basically to communication and motivation losses. Here we change it to Equation 5.1.

- Communication overhead

$$ComOverhead(n) = \begin{cases} \min\{\textit{Communication factor} * (n-1)^2, 100\} & n \geq 1 \\ 0 & n = 0 \end{cases} \tag{5.3}$$

Default value: *Communication factor* $= 0.05$

Explanation: Communication is an essential component in software development, but it is also an overhead. Within a bigger team, communication overhead among team members is obviously increased. Abdel-Hamid [1] lists several researches to show that it is widely held that communication overhead increases in proportion to n^2, where n is the size of the team as shown in Figure 5.2.

According to the graph by Abdel-Hamid [1], *Communication factor* is set to 0.05 as the default value. It can be calibrated later.

- Individual productivity

$$P_i = NomProductivity_t * f_{skill} * f_{experience} * f_{learning} * f_{overtime} * f_{schedulepressure} \tag{5.4}$$

Explanation: Individual productivity (P_i) is affected by certain factors, such as nominal productivity ($NomProductivity_t$), skill fitness (f_{skill}), the experience factor ($f_{experience}$), the learning factor ($f_{learning}$), the overtime factor ($f_{overtime}$), and the schedule pressure

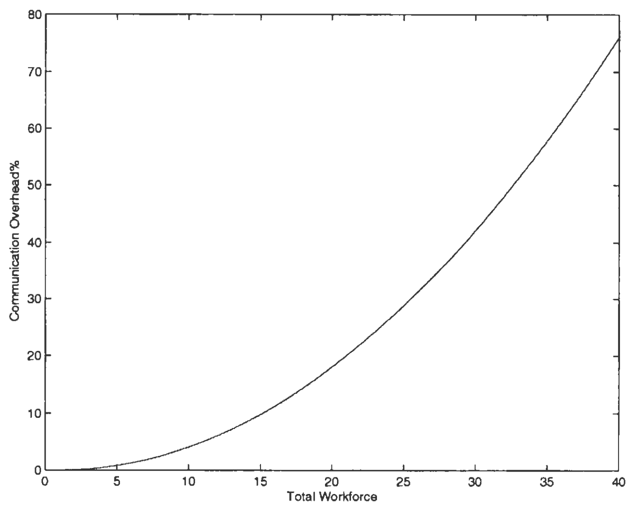

Figure 5.2 Relationship between workforce and communication overhead

factor ($f_{schedulepressure}$). Nominal productivity represents the maximum level of software development productivity that a certain task can be developed. Therefore, it is related to the task-related variable which can be retrieved from cases described in detail in Chapter 7. Table 5.2 shows an example of nominal productivity which we can retrieve from the database we set it before the simulation. It shows that if the task is within those properties (Task Type = 2, Complexity Level = 5), then we can get that Nominal Productivity = 0.8.

Table 5.2 A case of nominal productivity

variable name	*value*
Task Type	2
Complexity Level	5
Nominal Productivity	0.8

Other factors which are related to *individual productivity* will be discussed later in this section.

- Skill Fitness

$$f_{skill} = (\sum_{i=1}^{s} S_i/10)/s \tag{5.5}$$

Explanation: Same as timeline-model [17], the equation for f_{skill} is to calculate the skill fitness of an employee to a certain task. s is the number of skills required by the task. S_i is the proficiency of skill i. For example, Task 1 needs skill 1 and skill 3. Employee 1 has the skill 1 with a proficiency of 2 and skill 3 with a proficiency of 2. Employee 2 has skill 1 with a proficiency of 5 and skill 3 with a proficiency of 1. So Employee 2 has a higher skill fitness (i.e., 0.3) than that of employee 1 (i.e., 0.2).

- Experience

$$f_{experience} = adjustment_{experience} * E(tasktype_i, employee_j)/10 \tag{5.6}$$

Default value: $adjustment_{experience} = 1$

Explanation: $E(tasktype_i, employee_j)$ is the experience of Employee j on Tasktype i in software development, where $adjustment_{experience}$ is a factor to adjust the experience factor as it affects the capability dynamics.

- Learning

$$f_{learning} = \begin{cases} 1.6 * (l_i - 1) * X^2 + 1 & 0\% <= X <= 50\% \\ 1.6 * (l_i - 1) * X + 1.4 - 0.4 * l_i & 50\% <= X <= 75\% \\ l_i - 3.2 * (l_i - 1) * (1 - X)^2 & 70\% <= X <= 100\% \end{cases} \tag{5.7}$$

Explanation: X is the percentage that a task has been finished, whereas l_i is the learning property of an employee. Figure 5.3 shows an example of a learning curve with $l_i = 1.25$. This equation is adapted from a S-curve equation.

The learning curve has been studied for many years. Only a few papers, however, mention learning curve in Software Engineering, such as Raccoon [38]. Generally, learning curves are patterns which describe the long-term improvement in some stable processes, such as manufacturing and mining. Researchers have noticed that improved productivity increases year by year in those processes. Several patterns of the learning curve model exist: (1) The log-linear pattern of improvement: $y = a(x)^n$; (2) Standard-B: $y = a(x + b)^n$; (3)

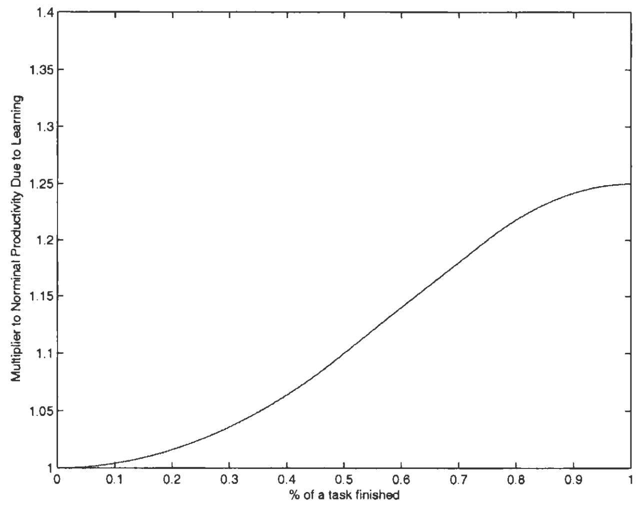

Figure 5.3 An example of learning curve in individual productivity

DeJong: $y = a + bx^n$; (4) S-Curve: $y = a + b(x + c)^n$. Each of these most popular models applies in a different field or has a different range of applications. It is observed that the log-linear equation can apply to a wide range of processes compared to the other three. Other equations also exist beside these four. Although there does not exist any concrete research to support these equations, we think they may fit in software development as a complicated stable process. Raccoon [38] argues that the learning curve can be applied in Software Engineering and analyzes the need to stabilize and improve the processes. In this case, an analysis is applied to the project staffing situation and one of the conclusions is to keep team members together on long-term projects. As a matter of fact, the definition of learning curve is not clear in the sense of different fields. Other papers discussing the learning curve in software process modeling indeed have different meanings, such as Hanakawa [23]. This paper proposes a simulation model for software development and tries to deal with software engineer's productivity efficiently. As to the problem of how we can model the learning curve in the software development process, we need to analyze real data and compare different models to know which one is more applicable.

- Schedule pressure

$$\text{schedule pressure} = \begin{cases} 5 & T_{current} > T_{idealfinishtime} \\ \min\{Workflow_{required}/Workflow_{normal}, 5\} & T_{current} \leq T_{idealfinishtime} \end{cases} \tag{5.8}$$

where

$$Workflow_{required} = effort_{remaining}/(T_{idealfinishtime} - T_{current}) \tag{5.9}$$

$$Workflow_{normal} = P_{team} \tag{5.10}$$

The lookup table for $f_{schedulepressure}$ is shown in Table 5.3.

Table 5.3 Lookup table for $f_{schedulepressure}$

schedule pressure (x)	$f_{schedulepressure}$
$0 \leq x < 1.1$	1
$1.1 \leq x < 1.35$	1.2
$1.35 \leq x < 1.75$	1.4
$1.75 \leq x < 3.5$	1.45
$3.5 \leq x \leq 5$	1.5

Explanation: When the human resources are constant for the project, it is impossible that people will do the job in the same rate when the deadline is near. Schedule pressure is a factor which can illustrate this situation. Because of the complexity of a human worker's psychology, we adopted simple relations between the schedule pressure and productivity (Table 5.1) from Vensim documents [47]. Equation 5.8, 5.9 and 5.10 are derived from the research by Abdel-Hamid [1] with revisions in the purpose of fitting our model.

Example: If an Employee's individual productivity is 100 LOC/day, then his productivity is 100*schedule pressure with the schedule pressure factor which will range from 100-200 LOC/day, 100 being without schedule pressure.

- Overworking

$$f_{overworking} = overtime * \text{fatigue productivity} \tag{5.11}$$

where overtime lookup table: (schedule pressure, overtime) = (1, 1), (1.2, 1.2), (1.5, 1.4), (2, 1.45), (5, 1.5), and

fatigue productivity lookup table: (overtime, fatigue productivity) = (1, 1) , (1.2, 0.9), (1.4, 0.75), (1.45, 0.6), (1.5, 0.3)

Explanation: From Vensim [47], we adopt the relationship of overtime and fatigue productivity lookup table in the example of project8.mdl.

Example: At task 2, the calculated schedule pressure is 2. So we look up the table and get overtime = 1.45. By looking at the fatigue productivity lookup table, we get the value 0.6. So it means the overall productivity is decreased from 1 to 0.6.

5.2.3 From System Meta Models to System Dynamics Simulation

Heuristic search provides a candidate scheduling solution from which we can perform the capability-based system dynamics simulation.

From the framework of our model in Figure 4.1, we can know that according to every candidate schedule, we can do the simulation to get the real task duration based on personnel/team capability.

For example, in a candidate solution such as that shown in Table 5.7.

- Employeel is assigned to taskl and the system dynamics simulation is shown as Figure 5.4.

- Employee 1 and employe2 are assigned to task 2 and the system dynamics simulation is shown as Figure 5.5.

5.3 Static Models

5.3.1 Employee Model

In the static model, an employee is represented by his ID and several static properties as shown in Table 5.4.

More explanation of these properties now follows.

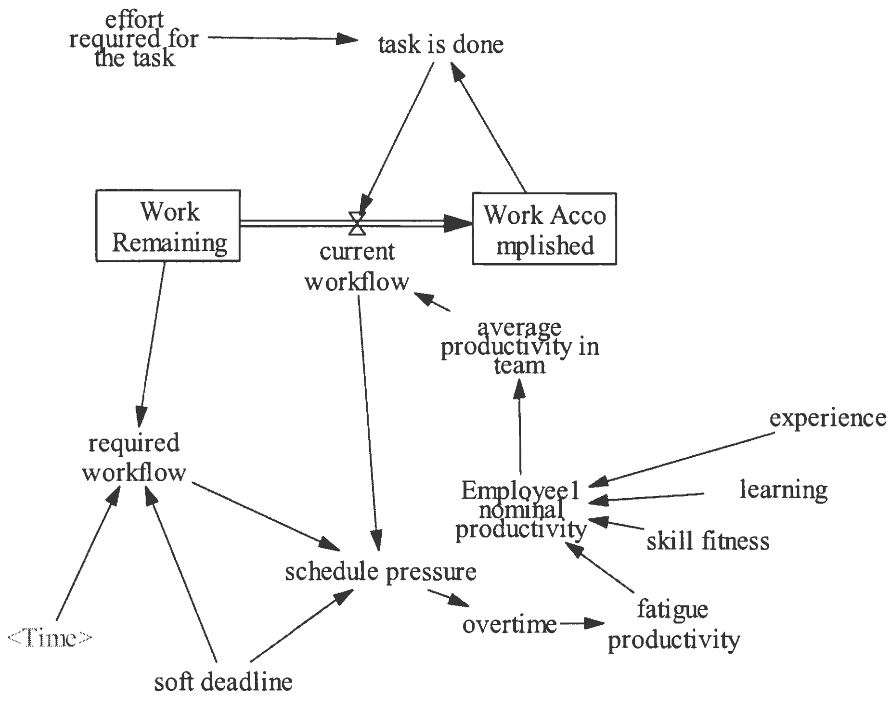

Figure 5.4 An example of system dynamics simulation (Task 1)

Table 5.4 Employee model

Property Name	*Type*	*Description*
Employee ID	Integer	a non-negative integer
Contractor	Boolean	True (contractor); False (not contractor)
Normal salary	Integer	monthly salary for 100% work load
Overwork salary	Integer	monthly salary for overtime load
Skill list with proficiency level	Array	a list of skills that this employee possesses
Learning factor	Double	factor to learning (≥ 1)
Max work load	Double	the upper limit for the work load of any employee; assumption: the maximum is 200%
Available start time	Integer	The available starting time for an employee
Available end time	Integer	the available ending time for an employee

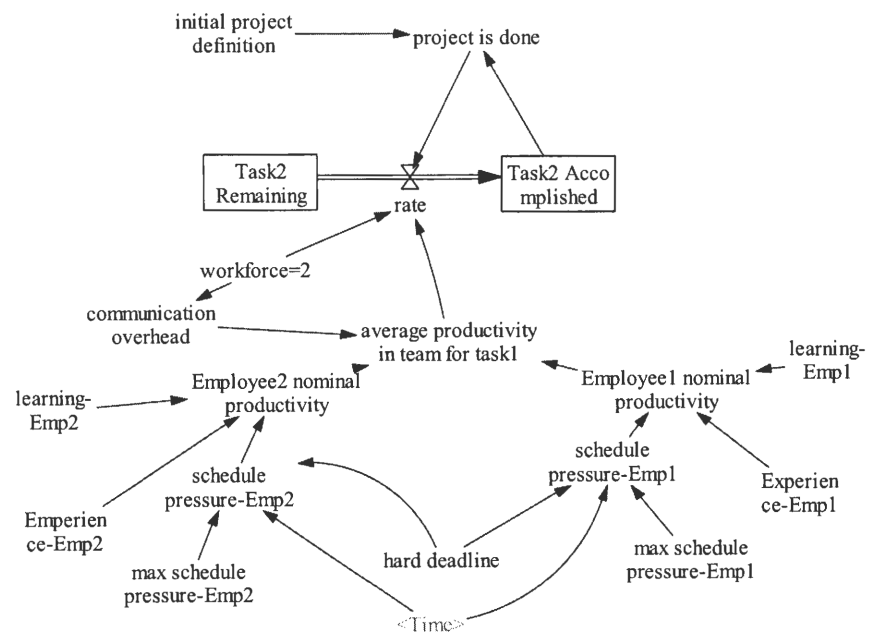

Figure 5.5 An example of system dynamics simulation (Task 2)

- Contractor or not: a contractor may not receive any training.
- Normal salary: the salary that the employees will get when they work a normal workload.
- Overwork salary: employees may be paid at a different rate when working overtime. Refer to Section 5.3.1.1.
- Skill list: a list with the skills that an employee possesses and a corresponding proficiency score (0-10) for each skill. 0 means the employee does not possess the skill and 10 means they are expert on it. Table 5.5 describes an example of an employee's skill list. Refer to Section 5.3.1.2.

Table 5.5 An example of skill list of an employee

Skill Id	*Skill Name*	*Proficiency*
1	c++	3.2
2	Perl	0
3	Java	4.5
4	VB	2

- Learning factor: this factor affects the learning curve described in Section 5.2.2.
- Max work load: it represents the maximum number of hours the employee can work. For example, 1.5 means this employee can work no more than 1.5 times his usual workload. On the basis of 40 hours per week, he cannot work more than 60 hours per week.
- Available start time and available end time: associated with each employee are two dates: available start time and available end time, which represent the months they first become and then cease to be available to a project.

5.3.1.1 Employee Payment Model

Employees may be paid at a different rate when working overtime.

- S_{normal} and $S_{overwork}$ are the salary rate when the employee works in normal work load and overwork load, respectively.

- h_{normal} and h_{max} are the normal working hours and maximum working hours for an employee, respectively.

The total payment for h hours is:

$$P(h) = \begin{cases} S_{normal} * h & 0 \leq h \leq h_{normal} \\ S_{normal} * h_{normal} + S_{overwork} * (h - h_{normal}) & h_{normal} < h \leq h_{max} \\ \infty & h_{max} < h \end{cases} \tag{5.12}$$

5.3.1.2 Skill Proficiency Acquiring Model

Skill proficiency can be gained by training (Section 5.3.2.2). If an employee's skill proficiency level on a given skill is r at the beginning of the training, at the end of training the skill is updated as,

$$Skill_{prof} = \min\{r + (LearningFactor - 1) * t, 10\} \tag{5.13}$$

LearningFactor refers to the property that an employee has. t is the duration of a training task. This equation is adopted from timeline-based model [17].

5.3.1.3 Experience Acquiring Model

Experience can be gained by doing similar tasks. If an employee's experience level on a type of task is r at the beginning of the task, then at the end of the task, his or her experience will change due to *learning*. The maximum value for experience is the same as that for proficiency, 10.0. Employee's experience of this task type after the completion of the task is updated as:

$$EXP(TaskType, EmployeeID) = \min\{r + (LearningFactor - 1) * t, 10\} \tag{5.14}$$

LearningFactor refers to the property of the employee. t is the duration of an employee assigned to the task.

5.3.2 Task Model

A task is represented as its ID and a set of properties. Table 5.6 shows those properties.

Table 5.6 Task model

Property Name	*Type*	*Description*
Task ID	Integer	the task's ID, a non-negative number
Description	String	description of the task
Task type	Integer	type of task (0-10)
Complexity level	Integer	(0-5) 0,1 means quite easy; 5 means very difficult
Ideal finished time	Date	If a task will be finished after this date, some penalty will be applied.
Hard deadline	Date	If a task cannot be finished before this date, the schedule is invalid.
Required skill list	Array	skill list required for finishing this task
Ancestors list	Array	a list of tasks that must precede this task
Required Effort	Double	nominal effort required to finish the task
Penalty Rate	Double	the penalty rate for a delayed task

These are the same as in the task-based model and, indeed, as in most scheduling models based on activity networks. A task still has a list of required skill ID's and a list of direct ancestor task ID's. All predecessor tasks must be completed before the task can begin.

5.3.2.1 Task Penalty Model

If a task is completed after the ideal finished time, a certain amount of penalty will be added to the total cost.

- T_{ideal} and $T_{deadline}$ are the ideal finished time and the hard deadline for a task, respectively.
- $PenaltyRate$ is the cost for finishing a task after its ideal finished time.
- T is the real completion time for a task.

The penalty cost for a task is:

$$TaskPenalty(T) = \begin{cases} 0 & T < T_{ideal} \\ PenaltyRate * (T - T_{ideal}) & T_{ideal} \leq T \leq T_{deadline} \\ \infty & T_{deadline} < T \end{cases} \tag{5.15}$$

5.3.2.2 Training Task

Training is considered as a special task in a project. Contractors may not receive any training. Training tasks have some of the properties that normal tasks have, such as *Task ID*, *Description*, and *Task Type* (which is 0 in this model). The differences are that the *required skill list* includes the skill that the employee is being trained in. Training tasks do not have any *penalty cost* or *required effort*, they do have *time duration.*

At the end of training, the skill proficiency of the employees assigned to training will be assessed and increased (refer to Section 5.3.1.2).

5.4 Problem Definition

For the scheduling problem under our definition, we have several assumptions:

(1) Each task has to be finished in a continuous duration.

(2) Different people can work on different tasks at the same time, but cannot do work over their maximum overwork level.

(3) Every employee assigned to the task needs to do the work in the whole duration for a certain task. It is not realistic for people to work on one task in one time unit and then switch to another job on the next time unit as stated in the timeline-based model [17].

5.4.1 Assignment Model

For each task, an employee can work with a load of 0%, 25%, 50%, 75% or 100%. One possible task-employee assignment to the example project in Figure 2.1 is shown in Table 5.7.

This scheduling means "Employee 1 can be assigned to do task 1 with 50% commitment, task 2 with 25% commitment, and task 4 with 25% commitment; Employee 2 does task 2 with

Table 5.7 An example of task assignment

	Task1	*Task2*	*Task3*	*Task4*	*Task5*	*Task6*
Employee 1	0.5	0.25	0	0.25	0.5	0
Employee 2	0	0.25	1	0.25	0.5	0.5

75% commitment, task 3 with 50% commitment, and task 4 with 25% commitment". Here 50% commitment means employee 1 can do 20 hours every week if he normally works 40 hours per week. So our assignment is like a task-based model with two dimensions (task, employee).

In order to reduce the computation burden in a timeline-based model, it will also be considered as a structure in our composite genome in later GA calculations.

5.4.2 The Objective Function

The objective function is used to determine the performance for each solution of the problem. It will be used later in Genetic Algorithm to select good individuals for the next generation. Different objective functions are possible for software project scheduling, e.g., minimum time to finish the project and maximum return on investment. Our main goal is to make the project overall as less costly as possible. So our objective function includes two parts mainly:

1. Validity of a schedule

A schedule is valid by checking both the validity of employee-task assignment and the validity of schedule on time. The criteria includes: all the skill needs of the tasks must be satisfied when assigning certain employee to the tasks; they must satisfy the precedence relations among tasks; all the tasks must appear in the schedule; no employee can work over maximum time limits at any point.

2. Minimum cost

We expect to have a schedule with a minimum total labor cost for performing the project achieved through using the cost of each resource, the duration applied to the tasks, and task penalties. During task execution, employees' capability can change, including experience on certain type of tasks and skill proficiency. This can affect the duration for a certain task. These properties have been discussed earlier in this chapter. Although there are other possible objectives that we may expect to achieve, we only focus on overall cost of a project at this stage. In the future, we can model more constraints and integrate the relevant parts to the objective function. The flexibility and power of using Genetic algorithm as an optimization technique represents its advantage.

5.4.3 Calculation of Fitness Function

The main steps to calculate fitness score are illustrated in Figure 5.6:

1) Initialize the system by loading a task-employee assignment

2) Set the number of the time unit

3) Get the next task from a topological sorted list

4) Check whether the task can start in this time unit or not by validating that all the precedence tasks have been finished, all the employees are available, and all the employees do not work over limit, if not, go to 2)

5) Do system dynamics simulation for each task

6) At the end of execution of every task, calculate the cost, penalty and update the employee's overall experience and certain skill proficiency

7) If all the tasks are finished, return the fitness score

8) Start another loop from 2)

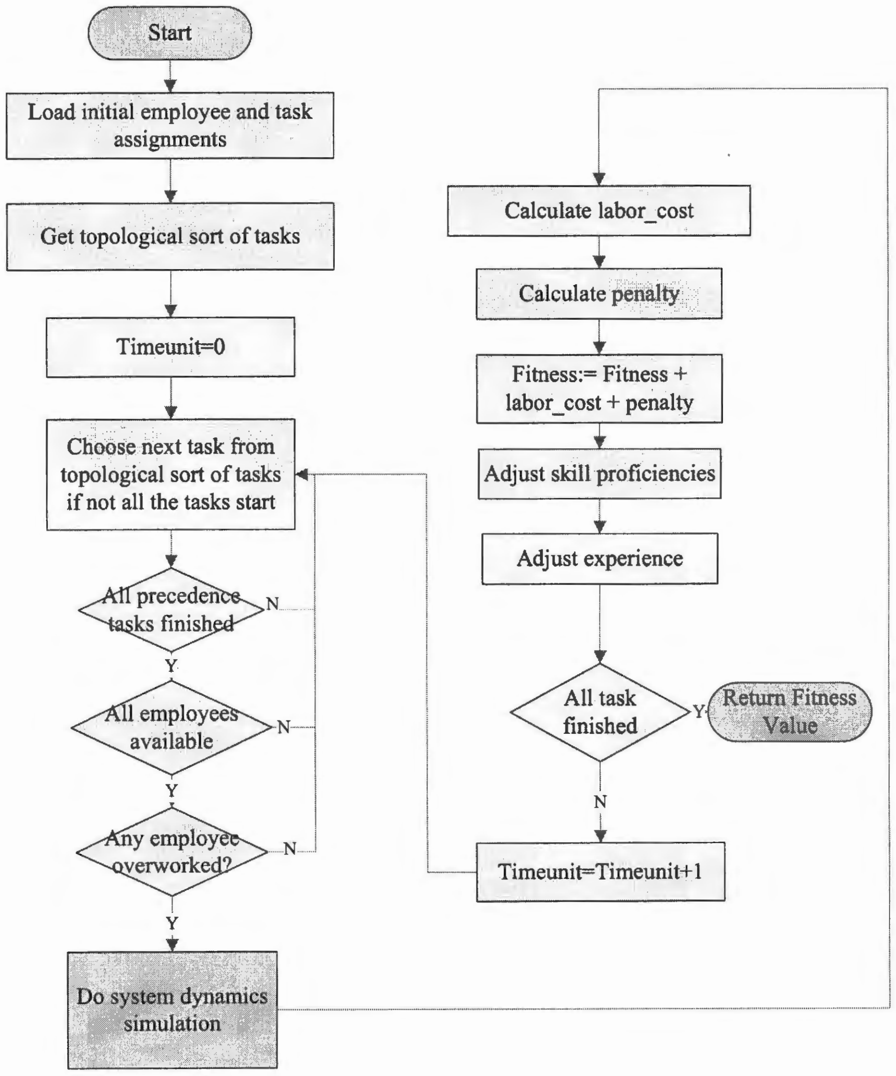
Start
Load initial employee and task assignments
Get topological sort of tasks
Timeunit=0
Choose next task from topological sort of tasks if not all the tasks start
All precedence tasks finished
All employees available
Any employee overworked?
Do system dynamics simulation
Calculate labor_cost
Calculate penalty
Fitness:= Fitness + labor_cost + penalty
Adjust skill proficiencies
Adjust experience
All task finished
Return Fitness Value
Timeunit=Timeunit+1
Y
N

Figure 5.6 Calculation of fitness function

CHAPTER 6. Scheduling with Heuristic Search

As introduced in Section 2.1.2, heuristic search has been widely applied in resource-constrained scheduling problems. The results of our previous work with GAs are reported [10], [17]. In this chapter, genetic algorithms for capability-based scheduling is described, such as genome representation and operators. The experimental results are presented. Additionally, the further result of the comparison of GAs and Hill-climbing on the timeline-based model is also reported.

6.1 Introduction to Genetic Algorithm

The main stages of genetic algorithms are shown in Figure 6.1. They begin with a group of initial solution individuals (a candidate solution of the problem) and execute iteratively to create better offsprings. The genetic algorithms apply genetic operators such as mutation and crossover to evolve the solutions until meeting some stopping criteria, for example, a certain number of generations have been generated, or the identity of the best individual has not changed for a number of consecutive generations. There are different kinds of GA algorithm.

Steady-state genetic algorithm: This algorithm is similar to the algorithms described by De Jong. It uses overlapping populations with a user-specifiable amount of overlap [18].

Simple genetic algorithm [21]: This algorithm uses non-overlapping populations.

Incremental genetic algorithm: This algorithm has overlapping populations with 1 or 2 children per generation.

Deme genetic algorithm: This genetic algorithm has multiple, independent populations. Each population evolves using a steady-state genetic algorithm with some individuals migrating from one population to another.

The three most important aspects of using genetic algorithms are: (1) definition and implementation of the genetic representation, (2) definition and implementation of the genetic operators, (3) definition of the objective function.

For the representation of individual genomes (collection of all chromosomes (blueprints) for an individual), Holland [26] worked primarily with strings of bits. There are other kinds of representation: arrays, trees, lists or any other object. Based on the representation, it is critical to define genetic operators (initialization, mutation, crossover, comparison). *Selection* of the parents and *crossover* (sometimes combined with *mutation*) are the construction of a child solution from the parent solutions. The selection process should choose individuals with better performance. A selection algorithm that gives little weight to performance will tend to search vastly but usually will not converge quickly. A *crossover* operator mimics the step to produce children inheriting certain traits of both parents. *Mutation* is a random process that is to randomly perturb some of the solutions in the population. In the absence of mutation, no child can ever acquire parametric values that were not already present in the previous population. Other than genetic operators, the *objective function* provides a measure of how good an individual is and it can be considered for either an individual in isolation or within the context of the entire population. The objective score is a measure used to evaluate the performance of the genome. There are many different variations to improve performance or parallelize the algorithms in recent research.

GAs are global search methods based on the evolutionary mechanisms and as a stochastic method, it has obvious advantages compared to random search and enumeration techniques [21], [32]:

1) The search procedure of genetic algorithm starts from a population, instead of a single starting point. This mechanism can help to jump off local optima, especially when applying strategies to keep the variety of the population.

2) During search procedure, the fitness score is used to evaluate a candidate solution, so no continuity of a function nor specific field knowledge are required compared to traditional optimization method.

3) When the problems are not continuous with multi-modality or noise, GA can still get

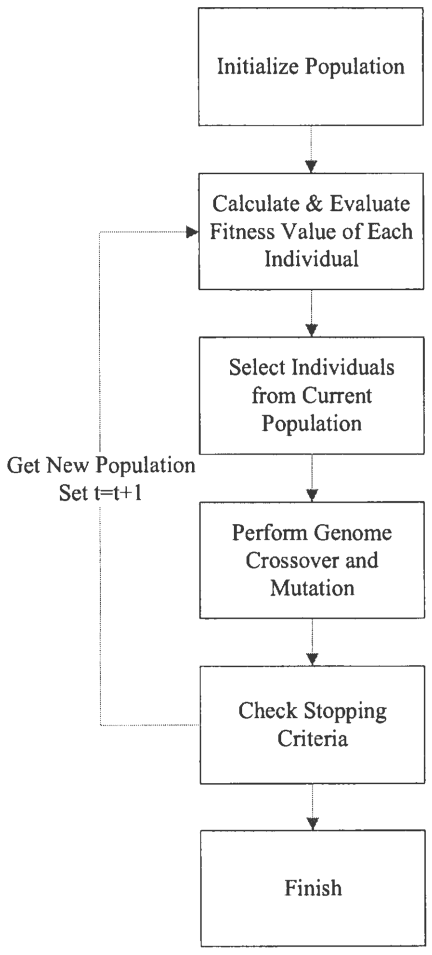
Initialize Population
Calculate & Evaluate Fitness Value of Each Individual
Select Individuals from Current Population
Perform Genome Crossover and Mutation
Check Stopping Criteria
Finish
Get New Population Set t=t+1

Figure 6.1 Main stages of genetic algorithms

near-optimal or satisfied solutions with high probabilities (robustness).

4) GA has the properties of implicit parallelism (dealing with the information of about O(n3) schemas in each generation), extendibility (easily adapted to other problems by changing the fitness function, or integrated with other algorithms, or adding more specific field knowledge) and scalability.

Most of research on stochastic modeling using Markov Chain proves that GA can converge to a global optimum with an infinite population size and infinite generation number [20], [41], [46]. But it is far away from actual application with time and space constraints. The other method to analyze GA is called "evolution dynamics", such as the building block hypothesis [21], a keystone of Genetic Algorithm approach. Building block hypothesis [21] says short schemata with high fitness are sampled exponentially more than other schemas and they are combined to form strings with expected higher performance. The equation below shows the relationship between the number of instances of a certain schema and its average fitness value:

$$m(H, t+1) = m(H, t) \times (f(H)/\bar{f}) \tag{6.1}$$

where H is a schema, $m(H, t+1)$ is the number of the instances of H at time $t+1$, $f(H)$ is the average fitness of all the instances of H at time t, $\bar{f}$ is the average fitness of the whole population at time t. If H is a schema with high fitness, the population of its instances will increase continually. Building block hypothesis guarantees that a problem can achieve to an optimum after searching the whole landscape. With a pilot study of scheduling problems in the project management, our problem involves many building blocks, that is, assigning the right person to the right task. Our problem favors the application of genetic algorithms.

The kind of problems GA cannot solve is an interesting topic. Many papers focus on the analysis of the relationship between the difficulty and modality of fitness landscapes [25]. Some research has explored attraction for the GA's operators. But no rigorous definition of the concept of difficulty is available in the framework of GAs [27]. Early research on characterizing difficulty has proposed criteria as isolation, deception, and multimodality. However, research shows that criteria of deception [49] and multimodality [25] do not work well. Another method requires the knowledge of the whole fitness landscape and is rather time consuming. It relates to the repartition of local optima around the global optimum [28].

Due to lack of any efficient and robust measure of difficulty and setting for a problem, experiments to try different settings and compare with each other are very popular since the early days of GAs as we did later in Section 6.3.1.

6.2 Genome Representation and Operators in Capability-based Model

6.2.1 Genome Representation

As we have seen, two dimensional array (with "Employee" enumerated along the rows, and "Task" along the columns) is used in task-based model [10] (refer to Section 3.1), whilst timeline-based model [17] uses a genome representation of three dimensional array (refer to Section 3.2). All the chosen encodings correspond to and are natural to each of the models respectively. The principle that using whatever encoding is the most natural to your problem and then devising a GA that can use that encoding has been widely accepted unless there is more theoretical progress on GAs [32].

According to the description of our capability-based model, we take out the time-line dimension but preserve other enhancements in our model as much as possible. We define our solution representation *SchedGenome* as a composite genome with two independent structures. It overcomes many of the complexities inherent in searches by not generating invalid solutions and causing no loss in expressiveness.

In the search space, any solution representation (S) is a mapping to a real schedule. For schedule, it includes two types of information: task-employee assignment and start time of tasks (the duration of a task can be achieved by system dynamics simulation). From task-employee assignment information, we have a one-to-one mapping from task-employee matrix to 1D task-employee array. Task start time is derived by topological sort of tasks according to certain priority list. Later, we will explain mappings in our genome representation.

Therefore, our solution representation of a schedule $S = \{A, L\}$:

- A is a 1D task-employee array that stores the information of task-employee assignments;

- L is the priority list by which a certain topological-sort vector representing the execution order of the tasks in the schedule can be derived.

An example solution representation with two genomes for Figure 2.1 is shown in Figure 6.2.

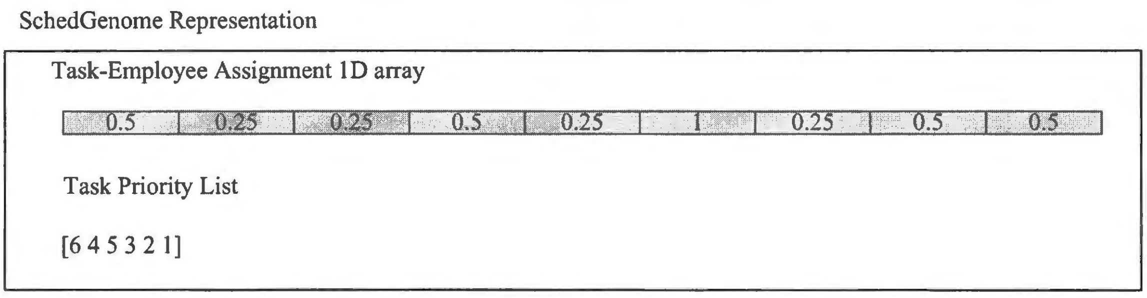

Figure 6.2 Illustration of the *SchedGenome* representation

6.2.1.1 Mapping Task-employee Assignment to 1D Array

The information of a task-employee assignment is "squeezed" into 1D array from original 2D array since 2D array may be a sparse matrix when the skill match of task and employee does not always happen. An example of the transition is shown in Figure 6.3. According to the skill match, we construct our possible-assignment matrix by setting the element (task to employee assignment) as 1 which is possible, otherwise 0. Then transit 2D array to 1D array by uniting all the rows in 2D array into a single row and at the same time delete the elements with value 0.

For matrix $A_{possible}$, $A_{possible}[t][i]$ is set to 1 if employee i can be assigned to task t. The elements of A_{2D} are stored in the 1D array genome A as shown in Function *Assignment2DTo1D*. Here is the pseudocode for this algorithm:

```
    $(A, A_{possible})$ Function $\textbf{Assignment2DTo1D}(A_{2D})$
1.  $size \leftarrow 0$
2.  for $i \leftarrow 1$ to TaskCount
```

Task-Employee Possible-Assignment Matrix

	Task1	*Task2*	*Task3*	*Task4*	*Task5*	*Task6*
Employee1	1	1	0	1	1	0
Employee2	0	1	1	1	1	1

Task-Employee Assignment 2D Array

	Task1	*Task2*	*Task3*	*Task4*	*Task5*	*Task6*
Employee1	0.5	0.25	0	0.25	0.5	0
Employee2	0	0.25	1	0.25	0.5	0.5

Task-Employee Assignment 1D Array

0.5	0.25	0.25	0.5	0.25	1	0.25	0.5	0.5

Figure 6.3 Illustration of 1D array structure in *SchedGenome*

```
3.  for $j \leftarrow 1$ to EmployeeCount
4.      if skill matched
5.          $A_{possible}[i][j] \leftarrow 1$
6.          $Size \leftarrow Size + 1$
6.          $A[Size] \leftarrow A_{2D}[i][j]$
7.      else
8.          $A_{possible}[i][j] \leftarrow 0$
```

6.2.1.2 Mapping 1D Priority List to Order of Task

The other genome is a list with information of task priority. For example, [6 4 5 2 3 1] is the task priority list for task 1 to task 6. We can see that task 1 with priority 6 and task 3 whose priority value is 5 has higher priority than task 2 with value 4. Then a topological sort can be generated to satisfy task precedence relationship.

Given a directed acyclic graph (DAG) $G = (V, E)$, a topological sort is an order of all the vertex and for each $(u, v) \in E$, u appears before v on the list. As we can see, each DAG may have more than one topological sort. Using task order as genome directly in our scheduling problem can generate invalid individuals. Another method, priority-based encoding, is proposed by Gen and Cheng [19]. Using priority-based encoding one can decide a certain order of tasks with information of tasks precedence information. When there are two tasks competing for one position, the task with the higher priority wins. Different priority can lead to different topological sort. Therefore, the encoding can represent all the possible topological sorts for a DAG.

The algorithm is to generate a topological sort by priority information. For each step, determine the set of eligible nodes; choose a node with highest priority and move it from eligible node set to partial topological sort list; update cut information of each unsorted node for the next iteration.

The pseudo code is as follows:

```
1.   for $i \leftarrow 0$ to $(taskcount - 1)$
2.       $pS[i] \leftarrow 0$ //the set of eligible node, i is in the set if $pS[i] = 1$
3.       $pCut[i] \leftarrow 0$ // the number of edges for vertex $i$ on the current cut
4.  for $i \leftarrow 0$ to $(taskcount - 1)$
5.       search for eligible nodes with $pCut[i] = d_{in}(i)$ and set $pS[i] = 1$
6.   if the node $u$ has max priority value
7.       $i \leftarrow$ index of $u$
8.       $pS[i] \leftarrow 0$
9.       put $u$ to sorted set
10.  for $i \leftarrow 0$ to $(taskcount - 1)$
11.      if $(u, v) \in E$
12.          $i \leftarrow$ index of $v$
13.          $pCut[i] \leftarrow pCut[i] + 1$
```

6.2.1.3 Mapping From Topological Sort to Start Time of Tasks

Given a topological sort, there could be more than one way to execute tasks. In our algorithm, we choose the earliest time that a task can start.

According to the fitness function calculation, start time of each task is determined from topological sort as shown in Figure 5.6.

6.2.2 GA operators

With our composite genome, we need to define the genetic operators to manipulate the two structures and need to account for the validity in the solution space. Since those two genomes have independent structure, the operators can be defined rather easily by manipulating one operator at one time.

6.2.2.1 Initialization

Choosing non-random initialization of the population is an important topic in scheduling. In general, it is true that using heuristics to choose better individuals for the initial population can lead to significantly faster convergence to a good solution. But initialization may have little effect on the performance of the solution if the evolution time is long enough. Our initialization operator randomly chooses any value from possible value (0, 0.25, 0.50, 0.75, 1) to be the allele (i.e., possible settings for an aspect of an individual) of the initial population. The list is initialized as the random order from 1 to the total number of tasks.

6.2.2.2 Comparison

The comparison operator is used to determine how one genome is different from another. The comparator thus provides a measure of how diverse a population is. The comparator of *SchedGenome* calls the comparators for each of the component genomes, then get the mean of the two scores. For the 1D array structure, given two solutions A and B, the distance between A and B is given by Equation 6.1. a_i and b_i are elements of A and B.

$$d = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2} \tag{6.2}$$

6.2.2.3 Mutation

The offspring may be mutated by the mutation operator at a rate determined by the mutation probability. Mutation is to modify one of the values used to encode the offspring. The function is intended to preserve the diversity of the population, thereby expanding the search space into regions that may contain better solutions. Mutation operators are also considered as background operator and will not affect much on the performance as crossover operators do.

```
$\{A_2, L_2\}$ Function $\textbf{\textit{Mutation}}$ ( $\{A, L\}$)
   //Randomly select one of the structures for crossover
1.  p ← a random real number in the range (0,1)
2.  If $p < 0.5$
3.     $A_2 \leftarrow mutation_A(A)$ //Mutation on the 1D array $A$
4.     return $\{A_2, L\}$
5.  else
6.     $L_2 \leftarrow mutation_L(L)$ //Mutation on the priority list $L$
7.     return $\{A, L_2\}$
```

According to the mutation probability, we randomly select one of the two genomes which is going to do the mutation shown in Figure 6.4. In the 1D array structure, we only change the certain elements while the certain elements are swapped in the task priority vector.

6.2.2.4 Crossover

The crossover operator mimics the way in which bi-sexual reproduction passes along each parent's good genes to the next generation. As to the crossover operator, the standard one-point crossover function is good at preserving short, good quality building blocks [32]. We choose one-point crossover for 1D array.

```
$\{A_{c1}, L_{c1}\}, \{A_{c2}, L_{c2}\}$ Function $\textbf{\textit{Crossover}}$ ( $\{A_{p1}, L_{p1}\}, \{A_{p2}, L_{p2}\}$)
```

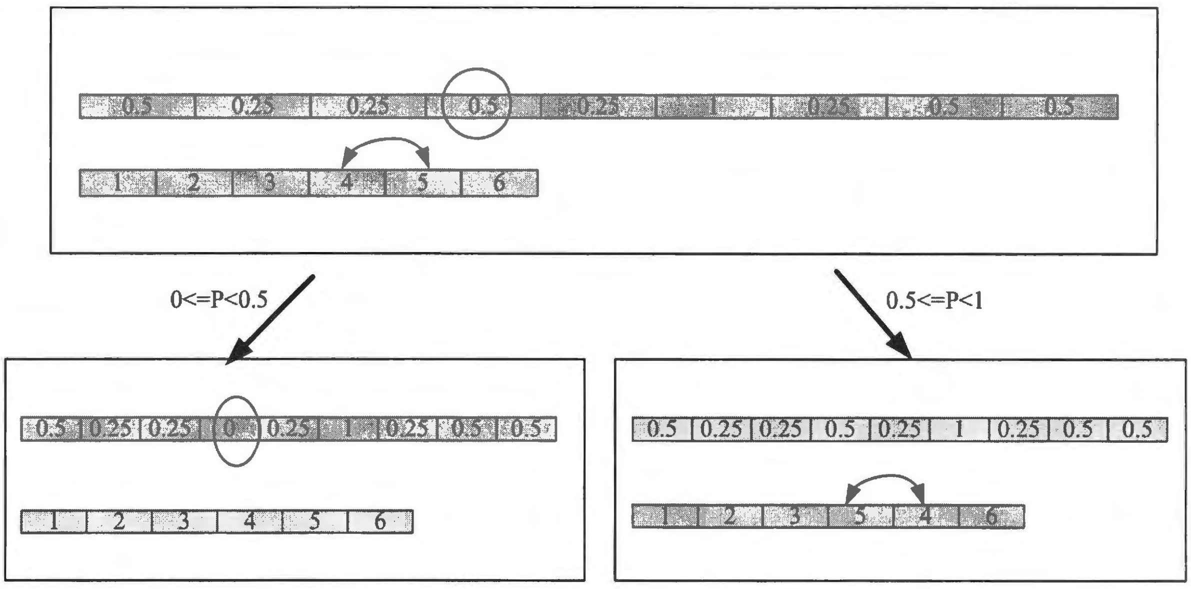

Figure 6.4 Illustration of *SchedGenome* mutation

```
    //Randomly select one of the structures for crossover
1.  p ← a random real number in the range [0,1)
2.  If $p < 0.5$
3.      $A_c \leftarrow crossover_A(A_{p1}, A_{p2})$ //Crossover the 1D array $A$
4.      return $\{A_c, L_{p1}\}, \{A_c, L_{p2}\}$
5.  else
6.      $L_c \leftarrow crossover_L(L_{p1}, L_{p2})$ //Crossover the priority list $L$
7.      return $\{A_{p1}, L_c\}, \{A_{p2}, L_c\}$
```

The crossover operator of *SchedGenome* invokes the crossover operator for each of the genomes in the composite genome according to a random number as illustrated in Figure 6.5. Mechanic simplicity facilitates relatively easy crossover on the task-employee 1D array as shown in the left part of Figure 6.5. By the crossover of the 1D array structures of parents, Child 1 and Child 2 are generated according to the crossover point of Parent 1 and Parent 2. Because the two structures of the genome representation are independent and the 1D array genome is derived from the possible-assignment matrix, the offsprings of the crossover operator are always valid.

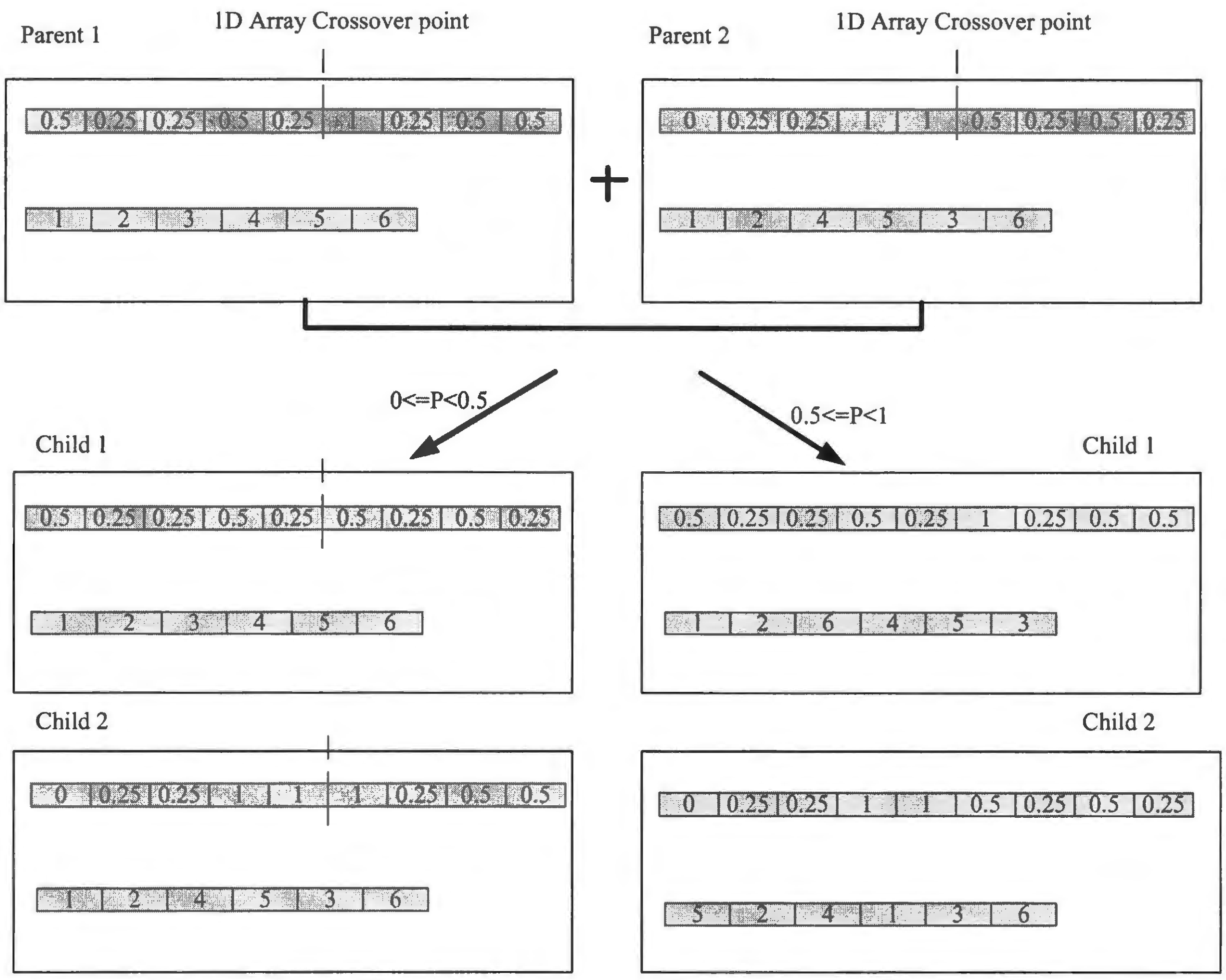

Figure 6.5 Illustration of *SchedGenome* crossover

For the list structure, the crossover operator should keep every offspring being an enumeration of their parents. For example, when parents are [123456] and [124536], each child should contain all of the elements in original list. The crossover preserves the ordering of elements by generating a random string with the same length as its parents. Child 1 copies from Parent 1 wherever the bit string contains a "0". The remaining strings of Child 1 have the same order as in Parent 2. Similarly, Child 2 copies from Parent 2 wherever the bit string contains a "1". The remaining strings of Child 2 have the same order as in Parent 1. An example of list-order crossover is shown in Figure 6.6.

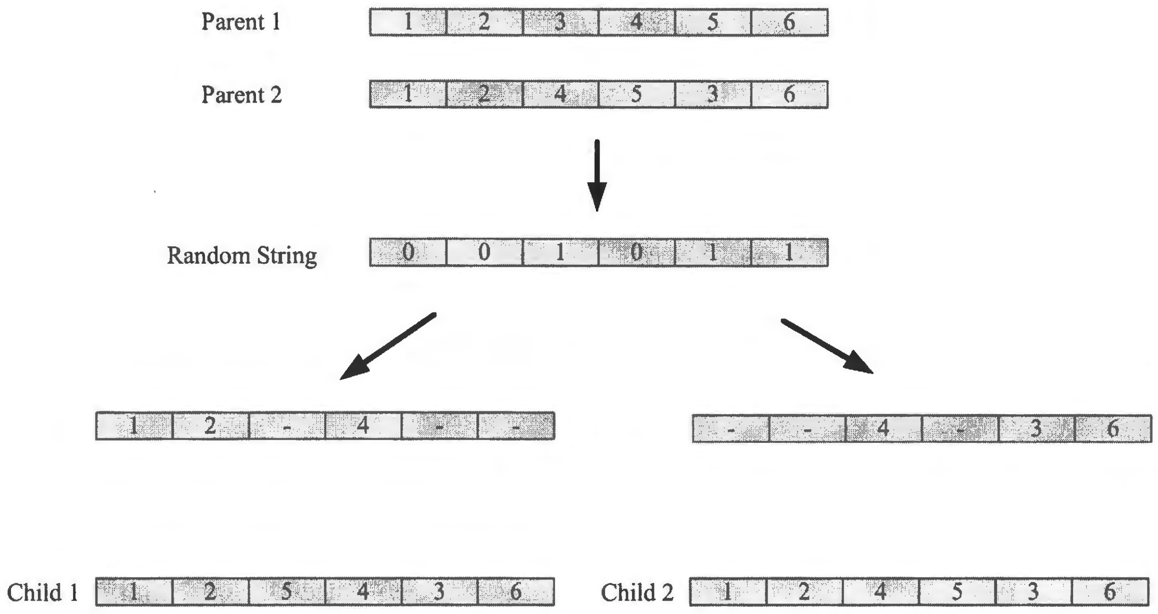

Figure 6.6 Illustration of list-order based crossover

6.2.3 Implementation in GALib

We use "Steady-state" algorithm (GASteadyStateGA) in GALib. The selection is Roulette Wheel. *2DArrayGenome* and *2DArrayAlleleGenome* classes provided by GAlib are implemented in task-based model and a three dimensional array (*GA3DArrayAlleleGenome*) is used in timeline-based model .

For our capability-based model, the task-to-employee assignment array that stores the information of task-employee assignments, and a topological-sort vector representing the execution order of the tasks in schedule are implemented by (*GA1DArrayAlleleGenome*) and (*GAListGenome*), respectively.

GA1DArrayAlleleGenome: The 1D array allele genome is derived from the 1D array genome class. It shares the same behaviors, but adds the features of allele sets. The value assumed by each element in an array allele genome depends upon the allele set specified for that element. Our single allele set includes all the possible values (0,0.25,0.5,0.75,1) to denote the percentage of time for an employee to be assigned a particular task.

Other than the genetic operators we defined by ourselves, we use the genetic operators provided by GAlib which are listed in Table 6.1.

Table 6.1 Genetic operators in *SchedGenome*

Genetic operators	*GAlib methods*
Comparison	GA1DArrayAlleleGenome::ElementComparator
Mutation	GA1DArrayAlleleGenome::FlipMutator, GAListGenome::SwapMutator
Crossover	GA1DArrayAlleleGenome::OnePointCrossover, GAListGenome::OrderCrossover

The crossover operator invokes the crossover for each of the genomes in the composite genome. Here we use the crossover operation for GA1DArrayAlleleGenome, OnePointCrossover, while OrderCrossover is used in GAListGenome. OrderCrossover is for keeping the length of the List genome.

6.3 Experimental Results of Capability-based Model

The case used in this section is shown in Figure 6.7. Employee properties and task properties are listed in Table 6.2, Table 6.3, and Table 6.4, respectively. There are several parameters that we can tune to get better results. For the preliminary parameter setting, the following 4 parameters are included:

- crossover probability
- mutation probability

- population size
- maximum size of generations

6.3.1 Preliminary Parameter Setting

Since genetic algorithms are non-deterministic, those factors, such as population size, generation number, mutation probability and crossover probability not only influence the time required to perform the GA algorithm but also affect the quality of the result. As to those parameters, they should not be too problem-specific because we may use them in different problems.

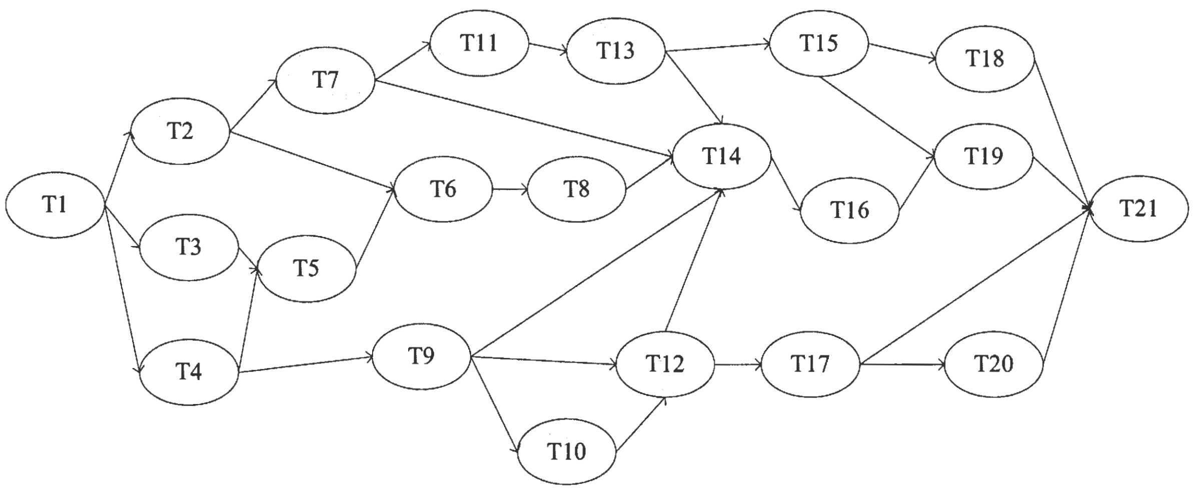

Figure 6.7 An example of test case

- Crossover Probability

To tune crossover probability, we set mutation probability, population number and generation number as 0.01, 1000, 1000 respectively. Crossover probability is set to 0.01, 0.05, 0.1, 0.4, 0.65 and 0.8 with the result shown in Figure 6.8. As can be seen from the figure, the influence of

Table 6.2 Employee properties

Emp ID	*Contract or Not*	*Basic Hourly Rate*	*Overwork Hourly Increase*	*Max Overwork Percentage*	*Initial Experience*	*Learning Factor*	*Start Date*	*End Date*
1	N	38	10	150	5	1.5	2005-01-01	2005-11-01
2	N	33	0	100	4	1.3	2005-01-01	2005-11-01
3	Y	30	5	125	3	1.3	2005-01-01	2005-11-01
4	N	35	0	75	4	1.1	2005-01-01	2005-11-01
5	N	35	5	125	4	1.1	2005-01-01	2005-11-01
6	N	36	10	125	4	1.4	2005-01-01	2005-11-01
7	N	36	0	10	4	1.5	2005-01-01	2005-11-01
8	N	35	0	100	4	1.1	2005-01-01	2005-11-01
9	Y	30	5	125	3	1.2	2005-01-01	2005-11-01
10	N	36	10	150	5	1.3	2005-01-01	2005-11-01

Table 6.3 Employee skill proficiency

Employee ID	*Skill 1*	*Skill 2*	*Skill 3*	*Skill 4*	*Skill 5*
1	4.5	4	0	5	5
2	0	0	2	0	5
3	4	4	2	0	5
4	0	4.7	3	3	0
5	4.5	4	4	0	5
6	0	4.5	4	4	0
7	4.5	5	5	0	0
8	0	0	3	4	5
9	0	0	4	5	0
10	5	4	0	3	4

Table 6.4 Task properties

Task ID	*Task Type*	*Complexity Level*	*Effort*	*Soft Deadline*	*Hard Deadline*	*Penalty Per Day*	*Skills Required*
1	1	3	0.25	2005-02-01	2005-03-01	1000	1 3
2	2	2	0.5	2005-03-01	2005-04-01	1000	2 3 5
3	2	5	0.8	2005-03-01	2005-04-01	2000	1 5
4	3	1	0.25	2005-03-01	2005-04-01	2000	1 2
5	4	3	0.6	0	0	0	1 3
6	2	2	0.5	0	0	0	3 4
7	3	5	0.3	0	0	0	2 4 5
8	2	1	0.4	0	0	0	2 3
9	5	2	0.25	0	0	0	2 4
10	4	4	0.5	0	0	0	4 5
11	4	4	0.5	0	0	0	1 3 5
12	4	4	0.25	2005-07-01	2005-08-01	1000	2 4
13	3	5	0.8	0	0	0	3 4
14	4	4	0.5	2005-09-01	2005-10-01	3000	1 2 3
15	3	4	1	0	0	0	2 3 4
16	4	4	0.5	0	0	0	4 5
17	4	4	0.5	0	0	0	1 3 5
18	4	4	0.25	0	0	0	2 4
19	3	5	0.8	0	0	0	3 4
20	6	4	0.5	0	0	0	1 2 5
21	7	5	1	2005-10-01	2005-11-01	5000	2 3 5

crossover probability does not seem to be that much. Later on, we will define our crossover probability as 0.65, as our previous works did.

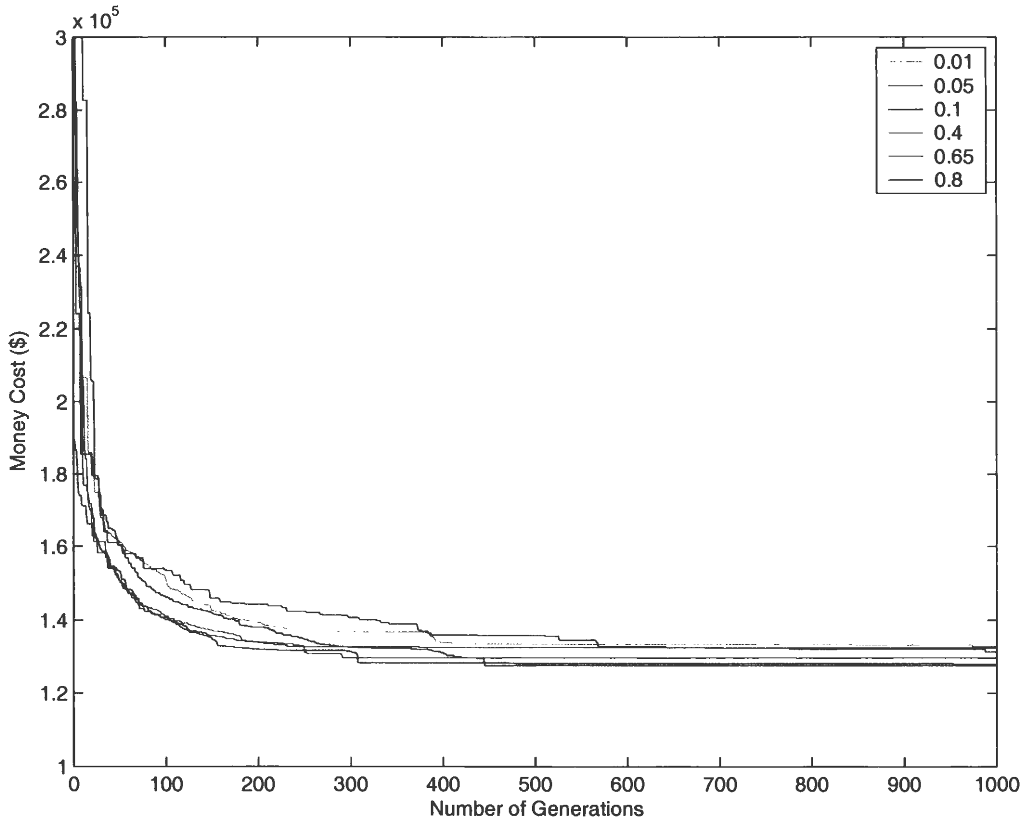

Figure 6.8 Comparison on crossover probability

- Mutation Probability

The influence of mutation probability on the results of the GA calculation must also be considered when tuning the algorithm.

From Figure 6.9, the comparison of results by different mutation probabilities is more meaningful. It suggests that small mutation probabilities produce better results than the larger ones do under our scenario. Such a phenomenon is not unique to this problem nor to the implementation [17]. This often occurs because higher mutation probabilities produce a greater percentage of not-so-good offspring. As the scheduling problem has many restrictions, it is easy to pro-

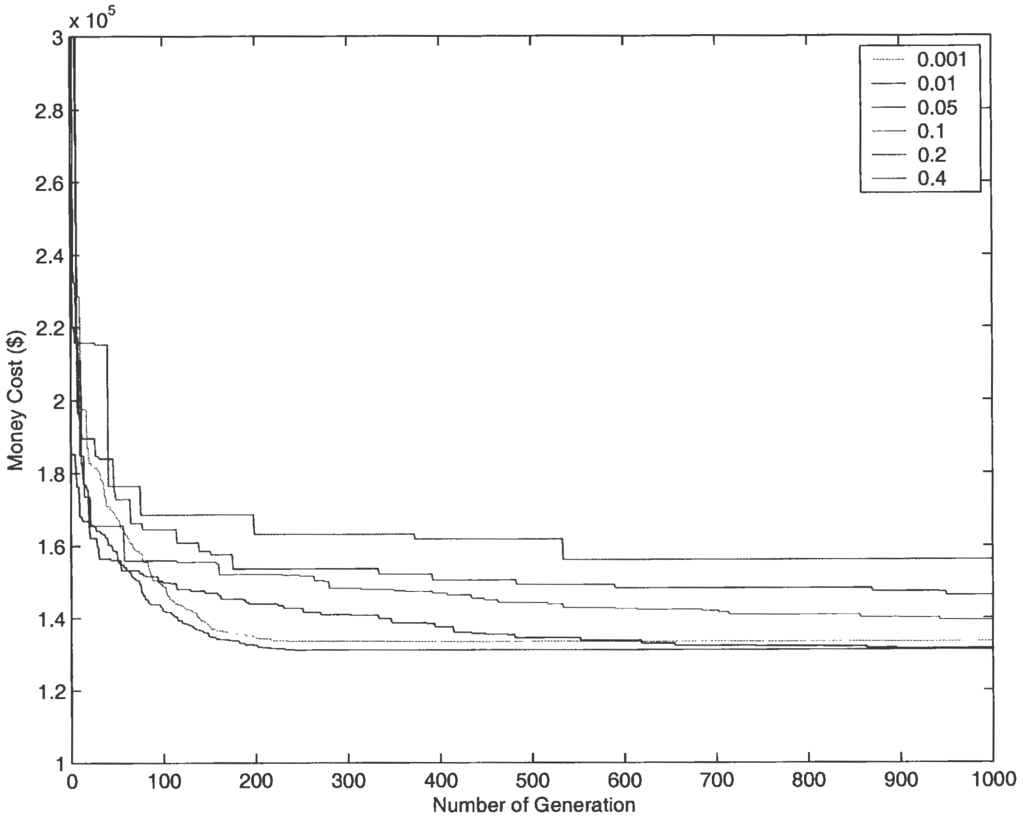

Figure 6.9 Comparison on mutation probability

duce such kind of not-so-good offsprings by random mutation. In the experiments reported herein mutation probabilities between 0.001 and 0.05 produced the best results. Accordingly, the default mutation probability is set to be 0.01 for the remainder of our work.

- Maximum Generation and Population Size

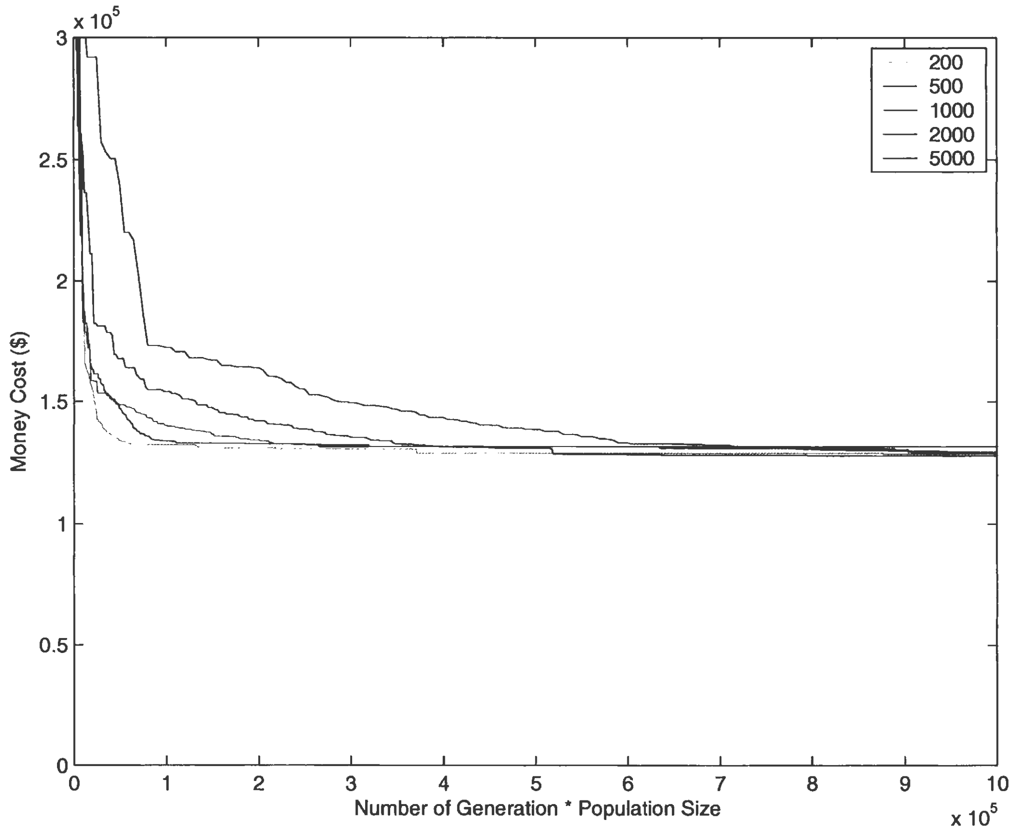

Figure 6.10 Balance on the numbers of generation and population

Although larger population size and generation number can definitely result in better performance overall, the balance of generation number and population size needs to take into account computation time. From Figure 6.10, after a certain amount of time, the performance with different parameters are close to each other. It is still reasonable to expect that those two parameters can be set to 1000 to get good results for larger experiments.

6.3.2 Results

After setting all the parameters, the best result from the search algorithm is shown in Figure 6.5, and the cost of the schedule is 127448, by considering all the factors. The Gantt chart for this schedule is illustrated in Figure 6.11.

Table 6.5 A near-optimal schedule from search

Task Name	*Duration*	*Start and End Date*	*Precedences*	*Resource*
Task1	11 days	Jan 1'05 - Jan 11 '05		Employee7
Task2	18 days	Jan 12 '05 - Jan 29 1 '05	1	Employee3[50%],Employee5
Task3	24 days	Jan 30 '05 - Feb 22 '05	1	Employee1,Employee3
Task4	16 days	Jan 12 '05 - Jan 27 '05	1	Employee1[50%]
Task5	21 days	Feb 23 '05 - Mar 15 '05	3,4	Employee3[25%],Employee7[75%]
Task6	15 days	Mar 17 '05 - Mar 31 '05	2,5	Employee6
Task7	22 days	Feb 23 '05 - Mar 16 '05	2	Employee1[50%]
Task8	11 days	Apr 3 '05 - Apr 13 '05	6	Employee7[75%]
Task9	5 days	Mar 17 '05 - Mar 21 '05	4	Employee1
Task10	12 days	Mar 22 '05 - Apr 2 '05	9	Employee1
Task11	18 days	Mar 17 '05 - Apr 3 '05	7	Employee3
Task12	10 days	Apr 3 '05 - Apr 12 '05	9,10	Employee1
Task13	29 days	Apr 4 '05 - May 2 '05	11	Employee6,Employee9[75%]
Task14	19 days	May 3 '05 - May 21 '05	7,8,9,12,13	Employee7
Task15	31 days	May 3 '05 - Jun 2 '05	13	Employee6
Task16	12 days	May 22 '05 - Jun 2 '05	14	Employee1
Task17	23 days	Jun 3 '05 - Jun 25 '05	12	Employee3[75%]
Task18	9 days	Jun 3 '05 - Jun 11 '05	15	Employee1[75%]
Task19	33 days	Jun 3 '05 - Jul 5 '05	15,16	Employee9
Task20	14 days	Jun 26 '05 - Jul 9 '05	17	Employee1
Task21	83 days	Jul 10 '05 - Sep 30 '05	17,20,19,18	Employee5

We also run tests by leaving our certain factors. From Table 6.6, we can see that without the learning factors, the overall cost increases. The cost without communication overhead decreases as expected. Without schedule pressure, no solution is found in this high constrained case. Although the results shown in the table are intuitive, in more complicated situation, the result can help analyze the influence of certain factors.

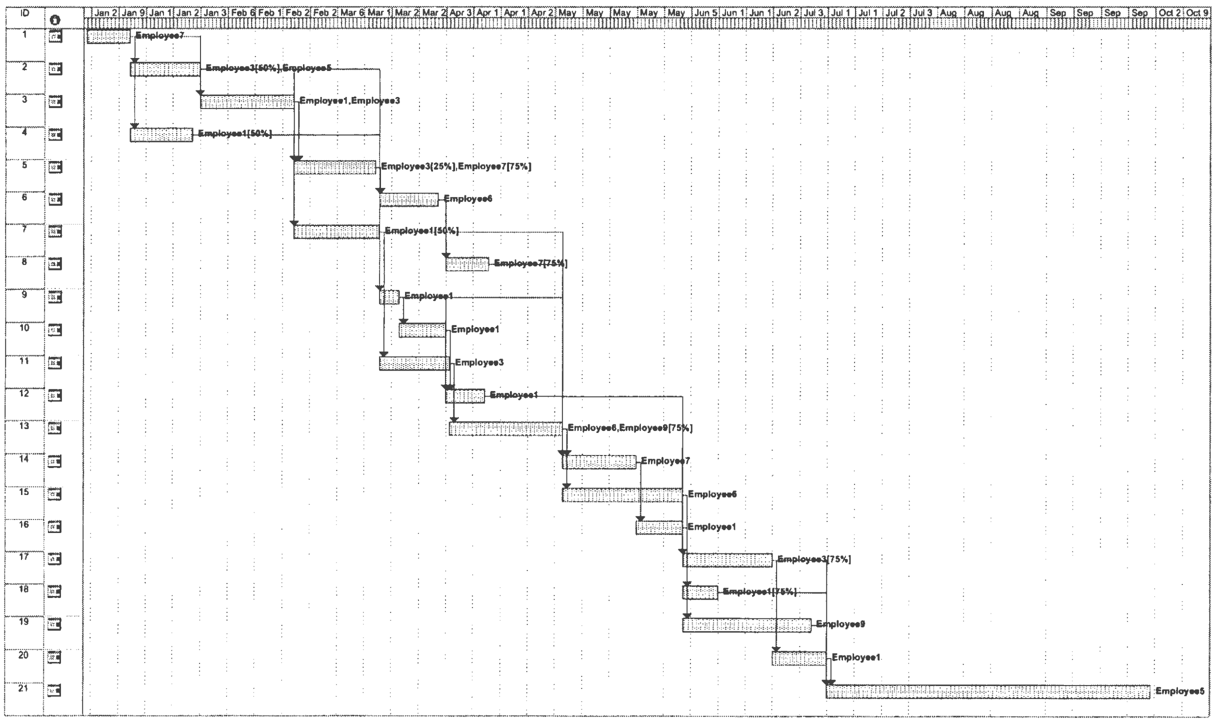

Figure 6.11 A schedule generated from test case

Table 6.6 Results without considering some factors

Cost	*Best*	*Worst*	*Average*
With all the factors	127448	135356	131104
Without learning factor	149426	152072	151174
Without communication overhead factor	125222	127478	126017

6.3.3 Discussion

Let us discuss the results from smaller experiments. Figure 6.12 shows a project with 4 tasks with task properties and employee properties in Table 6.7, 6.8 and 6.9.

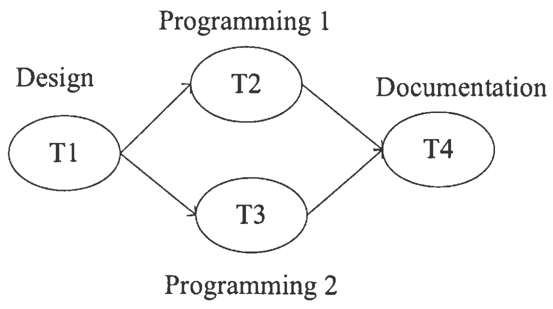

Figure 6.12 A test case with 4 tasks

Table 6.7 Employee properties

Emp ID	*Contract or Not*	*Basic Hourly Rate*	*Overwork Hourly Increase*	*Max Overwork Percentage*	*Initial Experience*	*Learning Factor*	*Start Date*	*End Date*
1	N	15	0	100	4	1.25	2005-01-01	2005-01-15
2	N	15	0	100	5	1.15	2005-01-01	2005-01-15
3	Y	18	0	100	5	1.45	2005-01-01	2005-01-15

Using same parameters from previous parameter tuning, we also get good results in this simple example, the result from GA is shown in Table 6.10. After the soft deadline is changed to Jan. 10 and the hard deadline is still kept at Jan. 15, we can see the result in Table 6.11 that employees are assigned to the work more to get things done more quickly. By comparing these two results, the difference is that Employee 1 is also assigned to Task 3 and the execution time for Task 3 decreases from 6 days to 3 days. It can decrease the penalty cost by finishing tasks earlier. Why is not employee also assigned to Task 1 to make the project done earlier?

Table 6.8 Employee skill proficiency

Employee ID	*Java*	*C++*	*Word*
1	5	3	2
2	4	4	4
3	2	5	2

Table 6.9 Task properties

Task ID	*Task Type*	*Complexity Level*	*Effort*	*Soft Dead-line*	*Hard Deadline*	*Penalty Per Day*	*Skills Required*
1	Design	4	0.1	2005-01-15	2005-01-15	20000	Java C++ Word
2	Programming	3	0.1	2005-01-15	2005-01-15	10000	Java C++
3	Programming	3	0.2	2005-01-15	2005-01-15	10000	Java C++
4	Documentation	3	0.2	2005-01-15	2005-01-15	10000	Word

Since even Employee 1 is added to the team for Task 1, the time to finish tasks can only be decreased a little bit but not enough to decrease it from 3 days to 2 days. Therefore, Employee 2 and Employee 3 are considered as the best solution for Task 1 in this situation. Why choose Employee 3 for Task 1 and Task 4 instead of Employee 1 when Employee 1 has higher skill proficiency with lower salary? It is because Employee 3 has 25% higher initial experience than Employee 1 and learning factor matters to a certain extent. From the above analysis, we can see the correctness of our model in some aspects.

We use same parameters in those tests. Although we get similar results, Table 6.12 shows a little different performance in different examples. In this relative big problem, the performance is best since all the results are close while results are in a wider range in relatively simple problems. Tuning GA parameters is not clear and our future work will focus on finding better rules to tune GA parameters for different categories of software project scheduling situations.

Table 6.10 A near-optimal schedule from search (soft deadline = Jan 15)

Task Name	*Duration*	*Start and End Date*	*Precedences*	*Resource*
Design	3 days	Jan 1 '05- Jan 3 '05		Employee2, Employee3
Programming 1	5 days	Jan 4 '05 - Jan 8 '05	1	Employee2
Programming 2	5 days	Jan 4 '05 - Jan 8 '05	1	Employee1, Employee3
Documentation	6 days	Jan 9 '05 - Jan 14 '05	2,3	Employee2, Employee3

Table 6.11 A near-optimal schedule from search (soft deadline = Jan 10)

Task Name	*Duration*	*Start and End Date*	*Precedences*	*Resource*
Design	3 days	Jan 1 '05- Jan 3 '05		Employee2, Employee3
Programming 1	5 days	Jan 4 '05 - Jan 8 '05	1	Employee2
Programming 2	5 days	Jan 4 '05 - Jan 8 '05	1	Employee1, Employee3
Documentation	3 days	Jan 9 '05 - Jan 12 '05	2,3	Employee1, Employee2, Employee3

Table 6.12 GA performance in a relative large problem (21 tasks, 10 employees) and small problem (4 tasks, 3 employees)

	Best	*Mean*	*Worst*
Relative big problem	127448	130311	136760
Relative simple problem	4296	5218	5914
Relative simple problem with more tight constraint	34248	39085	44200

6.4 Comparison of GA and Other Heuristic Search

To evaluate the performance of GA, an experiment on exhaustive search was conducted with the task-based model [10]. However, because the computational complexity of exhaustive search is extremely high in software project scheduling that comparison does not really show the performance of GA. No comparison was implemented among GA and any other heuristic search. So no evidence has really shown the power of GA or why GA is suitable for software project scheduling.

6.4.1 Comparison of GA and Hill-Climbing on Timeline-based Model

To evaluate the performance of a GA algorithm, the efficiency and quality of solving the problem needs to be known. There are two major criteria:

a) the number of function evaluations

We can compare the difference between the number of function evaluations needed in different algorithms to achieve the solution with the same quality or we can use the same number of function evaluations and compare the quality of a solution. It can be used for comparing different GAs or GA with other search algorithms.

b) On-line performance and off-line performance [14]

$$P_{on\_line}(s) = \frac{1}{n(T+1)} \sum_{t=0}^{T} \sum_{j=1}^{n} f(a_j, t) \tag{6.3}$$

$$P_{off\_line}(s) = \frac{1}{(T+1)} \sum_{t=0}^{T} f(a^*, t) \tag{6.4}$$

On-line performance reflects the change of the fitness values of the populations and shows the evolution of the whole procedure. Off-line performance reflects the best individual's evolution procedure and shows the search capability of GA.

Other than the two measures, Kallel [27] proposed some measures to characterize GA behaviors both on a temporal and spatial GA trajectory. They are based on Hamming fitness function for scaling.

A number of researches have defined certain kinds of problems that GAs work better than other heuristic methods and the criteria to compare different problems and algorithms [33], [25]. Mostly the comparison focuses on GA and Hill-Climbing. Here we try to show the performance of GA and hill-climbing on the timeline-based model based on the criteria of the quality of the best solution from different algorithm.

The hill-climbing algorithm that we use is that a population of initial solutions is chosen and the best one is chosen as the start point. By using that best one, it is mutated at a randomly chosen single locus and the fitness is evaluated. If the mutation leads to a higher fitness, the new one replaces the old one. The procedure continues until the optimum is found.

Usually hill-climbing algorithm is much faster than GA. However, the landscape in our problem has many local optima which makes hill-climbing difficult to achieve the global optimum.

An attempt for such comparison has been reported in our recent work [11]. The case consists of 15 tasks for which 10 employees were available. The employees, in turn, each possessed 5 skills to a greater or lesser extent. Each of the 5 skills was needed by at least one task and many tasks required multiple skills. The mean cost computed by hill-climbing is 34622840 with 1000 initial individuals in the population while the mean by GA is 27635360 which outperformed the best fitness achieved by hill-climbing as shown in Table 6.13, where lower number means lower cost and is better. Table 6.14 shows the comparison with a smaller case of 8 tasks, 7 employees in which GA outperforms HC dramatically, and Table 6.15 shows the comparison with a case of 3 tasks, 3 employees which do not show much difference in the two methods but the distribution of solutions from HC is obviously bigger than GA. Although these cases are random and more extended experiments need to be conducted, the results in general show the good performance and robustness of GA.

Table 6.13 Comparison between GA and HC (15 tasks, 10 employees)

	Best	Mean	Worst
Steady GA	25424896	27635360	29941800
Hill Climbing	29883000	34622840	41868400

In the early stage of the computation, because there exists many efficient, small blocks, under

Table 6.14 Comparison between GA and HC (8 tasks, 7 employees)

	Best	Mean	Worst
Steady GA	6645760	8021709	9401344
Hill Climbing	9695000	13099240	17308200

Table 6.15 Comparison between GA and HC (3 tasks, 3 employees)

	Best	Mean	Worst
Steady GA	8192	11298	15360
Hill Climbing	7428	11612	17422

crossover operators, the probability that the small blocks can be united as big blocks are high. Therefore, the quality of the population is improved quickly. But in the later stage, when big blocks are becoming more and more similar, the efficiency of crossover operators is becoming much lower. At that time the quality of many individuals cannot be improved a lot. In the later stage GA's efficiency is quite low.

By analyzing the computation procedure of GA, we can see from Figure 6.1(also refer to 6.8, 6.9, 6.10) that GA is not efficient in the later stage of computation which means it converges very slowly.

With GA's ability on global optimization and hill-climbing's ability on local optimization, combining GA with hill-climbing might be a good choice as a hybrid algorithm.

The tables below give the results from the experiment on two cases. Table 6.16 shows the result of GA (mutation probability = 0.001, generation number = 1000, population size = 1000) and GA (generation number = 500) with Hill-Climbing in a small experiment. Table 6.17 shows the result of GA and GA with Hill-Climbing in a relative larger project scheduling problem. Other cases are directed and show similar results.

In timeline-based model of software project scheduling, on relative smaller problems, hill-climbing is much better than GA both on time and fitness evaluations. On the other hand, GA outperforms hill-climbing on finding good quality solutions when the problems are becoming complicated, as is the case for many software projects, and in that case GA with hill-climbing does not help much in optimization. Therefore, GA is generally a good choice no matter the

Table 6.16 Comparison between GA and GA with HC in a relative small problem (3 tasks, 3 employees)

	Best	*Mean*	*Worst*
Steady GA	8192	11298	15360
Steady GA(500 generations) with HC	5286	6983	10024

Table 6.17 Comparison between GA and GA with HC in a relative big problem (15 tasks, 10 employees)

	Best	*Mean*	*Worst*
Steady GA	25424896	27635360	29941800
Steady GA(500 generations) with HC	26761000	28929233	33171600

problem size and the constraints in the software project scheduling problem.

CHAPTER 7. Conclusions and Future Work

This thesis proposes a framework for scheduling and monitoring in software project management. The genetic algorithm from previous work is improved by reducing the computation overhead without compromising with loss of realities. Experiments on comparing GA and the Hill-climbing method helped us determine that GA is a good method in software project scheduling. The automated optimization of scheduling can help a project manager for decision making by improving the quality of scheduling in software development.

However, there are several limitations of our framework. We can see that the capability-based scheduling framework can help make estimations and analyze the system dynamic behavior, especially in the following situations:

- When there is a lot of information about the personnel and organizational attributes, which can usually be obtained in the development stage;
- When the company collects project history data.

Therefore, our proposed work needs support from high quality software processes which can limit the application of the framework. Our capability-based model should be applied in a relatively high level of CMMI, and based on which we can do our quantitative calculation more realistically. By complementing the function of CMMI, Personal Software Process (PSP) and the Team Software Process (TSP) developed by Watt S. Humphrey give a way to measure an individual's development time and quality, which helps manage the teamwork planning and tracking thus providing more accurate data.

Another important aspect in project management is monitoring projects. Many failures of projects are due to poor monitoring. The feedback from actual project execution is so important in software project management that it can help improve estimation in later scheduling. For a certain project or a certain company, feedback can help calibrate the data and do re-scheduling when necessary. Although re-scheduling is very important, re-scheduling techniques are still not adequately explored in software engineering research.

Here we will briefly discuss the problem of calibrating our model, possible techniques to solve it and directions of our future work. Case-based reasoning (CBR) is a machine learning technique which is based on past experience. There have been some successful applications of CBR to cost estimation in software engineering [5], [16]. The case-based method in our system is designed to determine the parameters in the system dynamics part and to obtain the feedback from the real project implementation. To make these models more accurate according to certain projects and companies, we need to calibrate the uncertain parameters in these models. There are three different types of parameters: (1) known parameters; (2) unknown parameters; (3) imprecise parameters. Type 2 and 3 are the parameters that we need to tune. From a series of data, what we expect is to minimize the difference between estimated data and real data.

Illustrated by the simple example from Figure 2.1, suppose Task 1 has been finished and other tasks not, the calibration process is as follows. According to the first schedule, Employee 1 is assigned to Task 1 and Task 2; Employee 2 is assigned to Task 2. Suppose that Task 1 has been done (scheduled duration: 4 days. actual duration: 5 days) and other tasks have not started yet. Task 2 and Task 1 have the same properties on task type and complexity level. Additionally, Employee 1 and Employee 2 have the same experience level. Assume that "adjustment of experience" is the only factor that needs to be calibrated. From the difference between the actual duration and the scheduled duration of Task 1, the factor "adjustment of experience" is adjusted from 1 to 0.8. Assume that Task 2 is scheduled to finish in 6 days. Then after tuning the factor, the duration for Task 2 will be rescheduled to finish in 7.5 days.

For the re-scheduling problem, we will focus on those major concerns:

- Efficient algorithm

  For rescheduling, an efficient way will need to be found to favor the algorithm we have used, such as genetic algorithms. The goal is to ensure that the cost of rescheduling computation can be as small as possible.

- Performance tradeoff

  Efforts are needed to improve the rescheduling efficiency while maintaining the rescheduling quality. We need to do the risk analysis on the change of schedule. Tradeoff analysis between the performance gained by changing schedule and the cost required to change the schedule must be performed.

- Time to re-schedule partially completed tasks

  The appropriate time to do the re-scheduling is important. When there are some tasks that have already started in a project, a mechanism to automatically split partially completed projects is needed. Some method to looking ahead (for instance, to see if tasks are almost completed) is required. Then the projects need to split into two parts: one represents the completed part of the original project and the remaining tasks. To deal with partially completed tasks, more considerations should be added, such as whether re-scheduling of partially completed tasks is needed and how to do it.

Additionally, we suggest the following improvements to our work:

1. Modeling more aspects on software processes: currently we only focus on the development stage. Other stages, such as architecture design and testing can also be included in the framework.

2. Better modeling of the progress of tasks and employees: deeper and more comprehensive research is needed on the aspects affecting productivity, for example, learning and motivation. However, it requires interdisciplinary research among business, computer science, and psychology.

3. Model calibration: tuning system dynamics models is extremely important for accurate estimation. It includes the process of judging the validity of a system dynamics model, such as face validity (to test the fit between the rate/level/feedback structure of the model and the essential characteristics of the real system), reference mode replication (to test whether the model can reproduce the various reference behavior modes characterizing the system), extreme condi-

tion test (to test whether the model behaves reasonably under extreme conditions or extreme policies).

4. More comprehensive experiments for comparison and tuning: more experiments should be devised to analyze the performance of heuristic search algorithms. Rules for tuning GA parameters are still not clear.

5. Case studies on software project management: case studies are necessary and essential to evaluate the performance of our work. After the models have been completely established, case studies will help customize these models when needed. Research on integration with current commercial software tools can help transfer current advanced techniques such as what our research group developed into industrial use.

REFERENCES

[1] T. Abdel-Hamid, S.E. Madnick, *Software Project Dynamics: An Integrated Approach*, Prentice Hall, Englewood Cliffs, NJ, 1991.

[2] C. Andersson, L. Karlsson, J. Nedstam, M. Host, and B. Nilsson, "Understanding Software Processes through System Dynamics Simulation: A Case Study", *9th Annual IEEE Int'l Conf. and Workshop on the Eng. of Computer-Based Systems* (ECBS 2002), Sweden, 2002, pp.41-48.

[3] M. Barros, C. Werner, and G. Travassos, "System Dynamics Extension Modules for Software Process Modeling", *Workshop on Software Process Simulation Modeling* (ProSim '03), 2003; http://prosim.pdx.edu/prosim2003/paper/prosim03_barros.pdf (Date retrieved: June 2004).

[4] R. Bellman, *Dynamics Programming*, Princeton Press, Princeton, NJ, 1957.

[5] R. Bisio and F. Malabocchia, "Cost Estimation of Software Projects through Case-based Reasoning", *Proc. 1st Int'l Conf. Case Based Reasoning*, Sesimbra, Portugal, 1995, pp. 11-22.

[6] M.S. Boddy and R.P. Goldman, "Empirical Results on Scheduling and Dynamic backtracking", *Proc. 3rd Int'l Symp. Artifical Intelligence, Robotics and Automation for Space*, Pasadena, CA, 1994.

[7] B. Boehm et al., *Soft Cost Estimation with COCOMO II*, Prentice Hall PTR, 2000.

[8] P. Brucker, B. Jurisch, and B. Sievers, *A Branch and Bound Algorithm for the Job-shop Scheduling Problem*, Technical Report, Osnabrucker Schriften zur Mathemtic, Univ. Osnabruck, 1992.

[9] P. Brucker et al., "Resource-constrained Project Scheduling: Notation, Classification, Models, and Methods", *European Journal of Operational Research*, vol. 112, 1999, pp. 3-41.

[10] C. Chang, M. Christensen, and T. Zhang, "Genetic Algorithms for Project Management", *Annal of Software Eng.*, Kluwer Academic Publishers, vol. 11, 2001, pp. 107-139.

[11] C. Chang, Y. Di, and Y. Ge, "Time-line Based Model for Software Project Scheduling with Genetic Algorithms ", submitted to *Information and Software Technology* for review, 2004.

[12] C. Chao, *SPMNet: a New Methodology for Software Management*, doctoral dissertation, Univ. Illinois at Chicago, 1995.

[13] B. Clark, S. Devnani-Chulani, and B. Boehm, "Calibrating the COCOMO II Post-Architecture Model", *Proc. 20th Int'l Conf. Software Eng.* (ICSE), Kyoto, Japan, 1998, pp. 477-480.

[14] K.A. De Jong, *An analysis of the behavior of a class of genetic adaptive system*, doctoral dissertation, Univ. Michigan, 1975.

[15] L. Davis, "Job Shop Scheduling with Genetic Algorithms", *Proc. Int'l Conf. Genetic Algorithms and their Applications*, Pittsburgh, PA, 1995, pp.136-140.

[16] J.M. Desharnais, G.E. Wittig, G.R. Finnie,"Estimating Software Development Effort with Case-Based Reasoning", *Proc. 2nd Int'l Conf. Case Based Reasoning*, Providence, RI, 1997, pp.13-22.

[17] Y. Di, *Timeline Based Model For Job Scheduling with Genetic Algorithms*, master's thesis, Univ. Illinois at Chicago, 2001.

[18] GALib, "Lib: A C++ Library of Algorithm Components"; http://lancet.mit.edu/ga/(Date retrieved: June 2004).

[19] M. Gen and R. Cheng, *Genetic Algorithms and Engineering Optimization*, John Wiley & Sons, New York, NY, 2000.

[20] D.E. Goldberg, "Finite Markov Chain Analysis of Genetic Algorithm", *Proc. 2nd Int'l Conf. Genetic Algorithm* (ICGA 2), San Francisco, CA, 1987, pp.1-8.

[21] D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, MA, 1989.

[22] F. Glover, "Tabu Search - Part I. ORSA", *Journal on Computing*, vol. 1, 1989, pp. 190-206.

[23] N. Hanakawa, S. Morisaki, K. Matsumoto, "A Learning Curve Based Simulated Model for Software Development", *Proc. 20th Int'l Conf. Software Eng.* (ICSE), 1998, pp. 350-259.

[24] S. Hartmann, "A Competitive Genetic Algorithm for Resource-Constrained Project Scheduling", *Naval Research Logistics*, vol. 45, 1998, pp. 773-750.

[25] J. Horn and D.E. Goldberg, "Genetic Algorithms Difficulty and the Modality of Fitness Landscapes", *Foundations of Genetic Algorithms 3*, L.D. Whitley and M. D. Vose, eds., Morgan Kaufmann, 1995, pp. 243-269.

[26] H. Holland, *Adaptation in Natural and Artificial Systems*, Univ. Michigan Press, Ann Arbor, 1975.

[27] L. Kallel, "Inside GA Dynamics: Ground Basis for Comparison", *Proc. 4th Conf. Parallel Problems Solving from Nature*, 1998, pp. 57-66.

[28] S.A. Kauffman, "Adaptation on Rugged Fitness Landscape", *Lectures in the Sciences of Complexity*, volume I of SFI studies, Addison-Wesley, Reading, MA, 1989, pp. 619-712.

[29] S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi, "Optimization by Simulated Annealing", *Science*, vol. 220, 1983, pp. 671-680.

[30] P. Lakey, "A Hybrid Software Process Simulation Model for Project Management", *Workshop on Software Process Simulation Modeling* (ProSim '03), 2003;http://prosim.pdx.edu/prosim2003/paper/prosim03_lakey.pdf (Date retrieved: June 2004).

[31] S. McConnell, "Quantifying Soft Factors", Sept 2002; http://www.stevemcconnell.com/SoftFactors.pdf (Date retrieved: June 2004).

[32] M. Mitchell, *An Introduction to Genetic Algorithm*, the MIT Press, Cambridge, MA, 1996.

[33] M. Mitchell, J. Holland, and S. Forrest, "When Will a Genetic Algorithm Outperform Hill Climbing?", *Advances in Neural Information Processing Systems*, J. Cowan, G. Tesauro, and J. Alspector, eds., San Francisco, CA, 1994, pp. 51-58.

[34] A. Mingozzi, V. Maniezzo, S. Ricciardelli, and L. Bianco, "An Exact Algorithm for the Resource Constrained Project Scheduling Problem Based on a New Mathematical Formulation", *Management Science*, vol. 44, 1998, pp. 714-729.

[35] V. Plekhanova, "On Project Management Scheduling where Human Resource is a Critical Variable", *Proc. 6th European Workshop on Software Process Technology* (EWSPT-6), in *Lecture Notes in Computer Science series*, Springer-Verlag, London, UK, 1998, pp. 116-121.

[36] T. Potok, M. Vouk, A. Rindos, "Productivity Analysis of Object-oriented Software Developed in a Commercial Environment", *Software Practice and Experience*, vol. 29, no. 1, 1999, pp. 833-847.

[37] E. Ramat, G. Venturini, C. Lente, M. Slimane, "Solving the Multiple Resource Constrained Project Scheduling Problem with Hybrid Genetic Algorithm", *Proc. 7th Int'l Conf. on Genetic Algorithms*, San Mateo, CA, 1997, pp. 489-496.

[38] L.B.S. Raccoon, "A Learning Curve Primer for Software Engineers", *Software Engineering Notes*, vol. 21, no. 1, ACM SIGSOFT, Jan 1996, pp. 77-86.

[39] E. Roberts, "A Simple Model of R&D Project Dynamics", *Managerial Application of System Dynamics*, E. Roberts, ed., MIT Press, Cambridge, MA, 1981.

[40] I.R. Roman and M.R. Carrieira, "Dynamic Estimation Model for the Early Stages of a Software Project", *Workshop on Software Process Simulation Modeling*(ProSim '00), 2000;http://www.prosim.pdx.edu/prosim2000/paper/ProSimEA17.pdf (Date retrieved: June 2004).

[41] G. Rudolph, "Convergence analysis of canonical genetic algorithms", *IEEE trans. Neural Networks*, vol.5, no.1, 1994, pp.86-101.

[42] H. Sackman, W.J. Erikson, E.E. Grant, "Exploratory Experimental Studies Comparing Online and Offline Programming Performance", *Communications of the ACM*, vol. 11, no. 1, Jan. 1968, pp. 3-11.

[43] Standish group, "Extreme Chaos", survey report, 2001; http://www.standishgroup.com/sample_research/PDFpages/extreme_chaos.pdf (Date retrieved: Sept 2004).

[44] J. Sterman, *Business Dynamics: Systems Thinking and Modeling For a Complex World*, Irwin/McGraw-Hill, Boston, MA, 2000.

[45] C. Stevenson, *Software Engineering Productivity - A practical guide*, Chapman & Hall Computing, 1995.

[46] J. Suzuki, "A Markov Chain Analysis on Simple Genetic Algorithms", *IEEE trans Systems, Man and Cybernetics*, vol. 25, no. 4, 1995, pp. 650-659.

[47] Vensim Software Tool, Ventana Systems, Inc., 1996-2004; http://www.vensim.com/ (Date retrieved: June 2004).

[48] M. Wall, "A Genetic Algorithm for Resource-Constrained Scheduling", doctoral dissertation, Massachusetts Institute of Technology, 1996.

[49] S.W. Wilson, "GA-easy does not imply steepest-ascent optimizable", *Proc. 4th Int'l Conf. Genetic Algorithms*, San Diego, CA, 1991, pp. 85-89.

[50] B. Yang, J. Geunes, W.J. O'Brien, "Resource-Constrained Project Scheduling: Past Work and New Directions", Research Report 2001-6, Dept. of Industrial and Systems Engineering, Univ. Florida, 2001.

ACKNOWLEDGMENTS

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and the writing of this thesis.

First and foremost, I am indebted to my major professor in my graduate career: Dr. Carl Chang. I am grateful to him for his guidance, patience, and support throughout this research. He suggested me to improve the dynamic models of previous work and guided me to look into the theoretical basis of GA deeper. I learned many things (research, writing, cooperation skills, etc.) from Dr.Chang, whose vision and impeccable insights have been and will continue to be an inspiration to me.

I would also like to thank my committee member, Dr. Tsang Ming Jiang, for his valuable comments and suggestions such as conducting more experiments in comparison. I appreciate Dr. Daniel Berleant and Dr. Suraj C. Kothari for their help and serving as my POS members.

I would like to thank other members in our ICSE Lab, Hsin-yi Jiang, Jinchun Xia, Tae-hyung Kim, Dingding Lu, Jialin Le and Xia Wang. Without them, this work would not have been completed as smoothly and successfully as it has been.

Thanks also goes to Erica Bartsch, Susan Lee, and Curt Siemers for their generous help on checking my thesis. I appreciate my family group for their kindness and bringing me so much happy time. Thanks Feihong Wu for his valuable comments on my thesis writing.

Finally, I would also like to thank my family for the support they provided me through my entire life and in particular, I must acknowledge my mother, Huifen He, who stays with and takes care of me for more than 2 years in Ames. Thanks to my husband, Xin Wang, without whose love and support, I would not have finished this thesis.