A Simulation-Based Resource Optimization and Time Reduction Model Using Design Structure Matrix

by

Yifeng Zhang

B. Eng, National University of Singapore, 2007

Submitted to the School of Engineering
in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Computation for Design and Optimization

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2008

Signature of Author.....
Department of Computation for Design and Optimization
August 10, 2008

Certified by.....
Daniel E. Whitney, PhD
Senior Research Scientist – Center for Technology, Policy, & Industrial Development
Thesis Supervisor

Accepted by.....
Jaime Peraire
Professor of Aeronautics and Astronautics
Co-Director, Computation for Design and Optimization Program

A Simulation-Based Resource Optimization and Time Reduction Model Using Design Structure Matrix

by

Yifeng Zhang

Submitted to the School of Engineering
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computation for Design and Optimization

ABSTRACT

Project scheduling is an important research and application area in engineering management. Recent research in this area addresses resource constraints as well as stochastic durations. This thesis presents a simulation-based optimization model for solving resource-constrained product development project scheduling problems. The model uses *design structure matrix* (DSM) to represent the information exchange among various tasks of a project. Instead of a simple binary precedence relationship, DSM is able to quantify the extent of interactions as well. In particular, these interactions are characterized by rework probabilities, rework impacts and learning. As a result, modeling based on DSM allows iterations to take place. This stochastic characteristic is not well addressed in earlier literatures of project scheduling problems. Adding resource factors to DSM simulation is a relatively new topic. We not only model the constraints posed by resource requirements, but also explore the effect of allocating different amount of resources on iterations.

Genetic algorithm (GA) is chosen to optimize the model over a weighted sum of a set of heuristics. GA is known for its robustness in solving many types of problems. While the normal branch-and-bound method depends on problem specific information to generate tight bounds, GA requires virtually no information of the search space. Therefore GA makes this simulation-optimization model more general. Results are shown for several fictitious examples, each having some uniqueness in their DSM structure. Managerial insights are derived from the comparison of the GA solutions to these examples with other known solutions.

Thesis Supervisor: Daniel E. Whitney
Title: Senior Research Scientist, Center for Technology, Policy, and Industrial Development

ACKNOWLEDGMENTS

I would like to thank my thesis advisor Dr. Daniel Whitney, who has spent many hours with me, discussing the various topics involved in the project and answering my questions. He has always provided me with timely help and constructive feedback. He not only consistently gives me valuable advices from his perspective, but also encourages me to talk to other researches in the same field to get new ideas.

My sincere gratitude also goes to Prof. Tyson Browning and Prof. Rainer Kolisch, who have generously shared with me their research findings in DSM and project scheduling which are very helpful in my understanding of these two topics.

I would also like to thank Mr. Eric McGill, Mr. Jehanzeb Noor, Mr. Robert Corby, Ms. Tsoline Mikaelian, and Ms. Qi Hommes for helping me to get started with the DSM simulations, especially Eric and Jehanzeb, who have given me many valuable suggestions in the initial stages of my problem formulation.

A note of thanks must go to the administrative staff in the CDO office and the SMA office, especially Ms. Laura Koller, Mr. Michael Lim, Mr. John Desforge and Ms Jocelyn Sales.

A special thanks is owed to my academic advisor Prof. Robert Freund without whose encouragement and help this thesis would not be possible.

Last but not least, I would like to thank my family and friends for always supporting me and caring for me.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

Intense global competition, rapid technological advances and changing patterns of the world market opportunities compel companies to put a lot of effort in the research of product development processes recently. The complexity of a typical product development process is first demonstrated by its sheer number of activities involved. Moreover, the interactions among the activities are complicated. The exchange of design information are normally non-unidirectional. There are inherently many iteration loops hidden in the process. To accurately model the interactions involved in a product development process, to understand the impact of these interactions as well as to schedule the activities under constraints, most commonly resource constraints, are the keys to success.

In this thesis, we treat each design process as a project, in which a set of interdependent tasks need to be executed. The *Design Structure Matrix* (DSM) is chosen to represent these tasks and their interactions. DSM has several advantages over other project representation/analysis tools, as will be discussed in detail in later chapters. Firstly, it provides the user with a concise graphical representation of the process. Moreover, interactions are quantified in terms of rework probabilities, rework impacts and learning effects. As a result, the effect of iterations becomes visible and can be studied in greater depth. Last but not least, it eases modifications such as re-sequencing the tasks: minimal changes are needed to obtain the new structure.

Modeling through DSM allows dynamic analysis of the product development process. One major attribute that we are interested in is the total process duration, since whether a company is able to launch a new product on time is crucial. We will study the effect of resource constraints and reworks on process duration, as well as the relationship between resource and reworks.

Combining simulation with optimization has received more and more attention in recent literatures [2]. This is a result of improving computing power, as well as people's increasing interest in studying stochastic complex systems. *Genetic algorithm* (GA) is used to optimize the resource-constrained scheduling problem in the product development process. Numerical

examples show that it is indeed a robust method, which performs well for a large variety of problems, and is able to overcome difficulties faced by many non-heuristic optimization methods. The rest of the report is organized as follows. In Chapter 2, some background knowledge of DSM is introduced. The advantages of using DSM over other representation tools are discussed. Chapter 3 presents the dynamic modeling using DSM. The simulation algorithm is outlined. A short literature review of resource-constrained scheduling problems is presented in Chapter 4, followed by a detailed description of the implementation of GA in this particular model. In Chapter 5, results of numerical examples are analyzed and discussed, followed by concluding remarks in Chapter 6.

CHAPTER 2
DESIGN STRUCTURE MATRIX REPRESENTATION AND MODELING

Design Structure Matrix (DSM) is also known as Dependency Structure Matrix, Problem Solving Matrix or Design Precedence Matrix. It is a system analysis tool which provides a compact and clear representation of a complex system. It captures the interactions/interdependencies/interfaces between system elements. It is also a project management tool which provides a project representation that allows for feedback and cyclic task dependencies. [21]

2.1 Basic DSM Components

There are four types of DSMs, namely component-based, team-based, task-based and parameter-based DSMs. The difference between them is the type of data they represent. As a result, subsequent treatments of the DSMs are also different. In this thesis we would like to capture and study the input/output relationships among tasks of a project and hence gain insights in task sequencing, time reduction and ultimately project scheduling, so we are only concerned with task-based DSMs.

A binary DSM is the most basic form of DSMs. It is a square matrix consisting of "1"s (or marked with "×"s) and "0"s (or left blank). While a "1" represents some kind of interaction, a "0" means no interaction is present. Figure 2-1 shows an example of a binary DSM constructed for a 7-task project [21]. The names of the tasks A, B, ...G are marked across the rows and the columns of the matrix, representing the tasks in the same order. The diagonal entries have no significance, and hence are left blank.

Reading across a row reveals the inputs to an task. For example in the first row, there is a mark each in the cells (A,C) and (A,D). This means task A depends on tasks C and D; their outputs are the inputs to task A. In the third row, four marks appear in cells (C,A), (C,B), (C,F), and (C,G),

meaning task C needs information from tasks A, B, F, and G as input. On the other hand, reading across a column reveals the output flow of an task. In this example, output of task A flows to task C; output of task C is needed by tasks A and E.

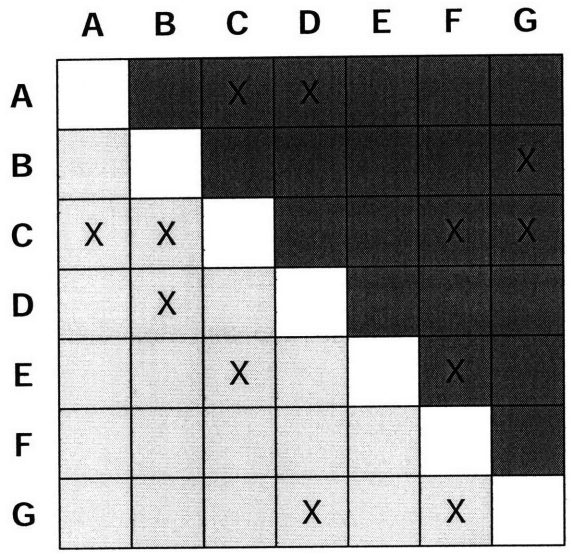


Figure 2-1 A binary DSM representation of a 7-task project

Even though all the marks represent interactions, if we assume the tasks are carried out in the sequence shown in the DSM, the marks below and above the main diagonal have different significance: a sub-diagonal mark represents information feed forward from an upstream task to a downstream task, whereas a super-diagonal mark represents information feedback from a downstream task to an upstream task.

The "×" in cell (C,A) is a feed forward mark, representing output of task A flowing to a downstream task C. Since A is finished before C. The information needed by C from A is always available. Therefore feed forward marks are desirable. On the other hand, feedback marks are undesirable. For example, the "×" in cell (C,G) represents information feedback. C uses G's output as part of its input. However, when C is executed, G is not started yet, so it cannot provide C with the necessary input. C has to be executed with partial information by making assumptions or estimations of G. When G is done, the assumptions and estimations made about G earlier needs to be checked against the actual output of G. If there are significant discrepancies, C has to

be reworked. Therefore we seek to minimize the number of feedback marks in order to reduce the amount of rework (iterations), hence to shorten the total project duration.

2.2 Advantages of DSM Representation and Modeling

2.2.1 A Concise Graphical Analysis Tool

DSM provides a concise graphical representation of a project. We shall compare it with a diagraph, which is commonly seen in project scheduling problems. Figure 2-2 is a digraph representation of the same project we have discussed in the previous section. The strength of DSM representation is obvious. The digraph looks pretty complicated even for such a small project. When the project involves tens or hundreds of tasks, which is not uncommon, the digraph representation is definitely not a good choice, since the jumbled arcs do not help systematic thinking. On the other hand the DSM representation enables us to easily observe and quantify the extent of interactions. The number of marks per row and per column is one metric for project complexity. Moreover the distribution of interactions can also be visualized. For a second example shown in Figure 2-3, it is clear that there are at least four major iteration loops in the DSM. It would not be obvious if the same project is represented by a digraph.

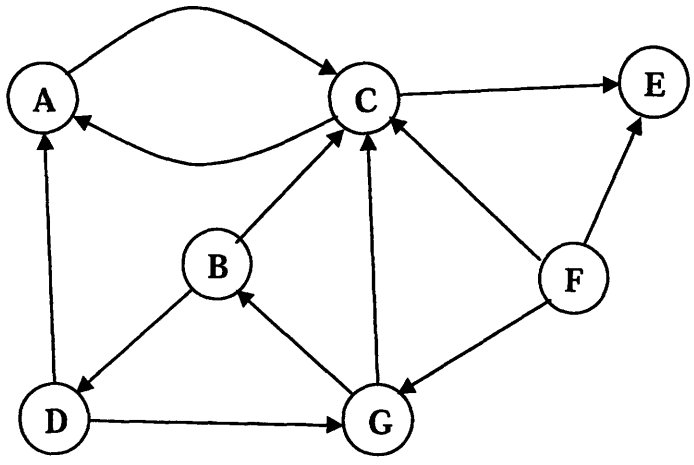


Figure 2-2 Digraph representation of the DSM in Figure 2-1

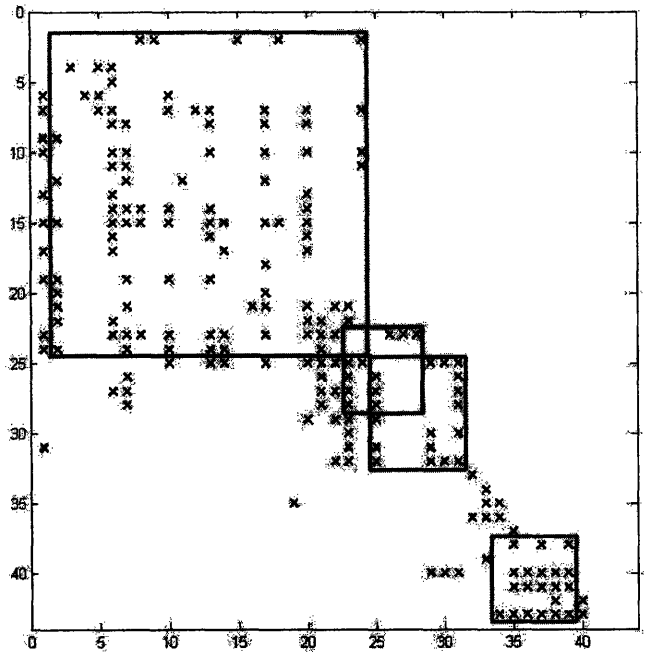


Figure 2-3 Iteration loops in a project

2.2.2 A Multitude of Attributes

In a binary DSM, only one attribute is used to show the relationship between different tasks: the existence or the absence of a dependence relationship. Compared to binary DSMs, numerical DSMs can contain a multitude of attributes, which give greater details of the relationships between the tasks [21]. An improved sophistication of data capture and representation provides better understanding of the project, and allows for better optimization algorithms to be developed. The attributes considered in this thesis, and which are not normally included in other project scheduling problems, include stochastic task durations, rework probabilities, rework impacts and learning curve.

The diagonal entries in a binary DSM have no significance, whereas a numerical DSM's main diagonal can be used to store the nominal values of task durations. They are normally the *most likely values* (MLV) of the durations needed to finish the tasks of a project. Apart from the nominal values, *best case values* (BCV) and *worse case values* (WCV) are also included. They

can appear as extra columns appended to the main DSM. Knowing these three values, we can construct distributions from which the task durations can be sampled stochastically.

Task sensitivity (TS) describes how sensitive the completion of a dependent task is to changes or modifications of information of an input task. Information variability (IV) measures the likelihood that information provided by an input task would change after being initially released. These two attributes are closely related but inherently independent. Their product is called task volatility (TV):

$$TV = TS \times IV$$

TV is a property that represents the robustness of dependent tasks with respect to the changes in information from input tasks. It gives us a sense of the probability that information feedback will occur, which typically results in rework. Therefore through proper scaling, these three attributes can be eventually condensed into one quantity, which is the rework probability. Rework probability takes values in the range of [0, 1]. It is the probability that a dependent task needs to be redone when changes occur to an input task. In a numerical DSM, these rework probabilities replace the "1"s or the "×"s of a binary DSM.

While TV measures the probability of information feedback, and hence rework probability, it does not indicate the level of impact that the feedback has on the task that receive it. Therefore an additional attribute, rework impact, is used to characterize this aspect of rework. It also ranges between 0 and 1. It represents the percentage of the original work that needs to be reworked. Every rework impact value corresponds to one rework probability in the DSM. They appear in the 2nd plane of the numerical DSM.

Learning curve is a property that represents the amount of time required, as a percentage of the original duration of the intended work, when the task is done subsequent times. The values are between 0 and 1. They also appear as an additional column in a numerical DSM. [27]

With these attributes added to the DSM, the project model fidelity has greatly increased.

2.2.3 Ease of Modification

As discussed in section 2.1, feedbacks result in potential rework, which is undesirable. Therefore re-sequencing is necessary to obtain the best task execution sequence, so that the amount of feedback and rework is minimized. In terms of DSM operation, it is equivalent to shifting the rows and columns of the DSM to reduce the number of super-diagonal entries. Two special DSM operations have been proposed and studied recently, namely partition and tearing.

Partition refers to the process of manipulating (i.e. reordering) the DSM rows and columns such that the new DSM arrangement does not contain any feedback marks, thus, transforming the DSM into a lower triangular form. For complex systems, it is highly unlikely that simple row and column manipulation will result in a lower triangular form. Therefore, the analyst's objective changes from eliminating the feedback marks to moving them as close as possible to the diagonal (this form of the matrix is known as block triangular). In doing so, fewer tasks will be involved in the iteration cycle resulting in a faster development process. [21]

Using the same example in section 2.1, the DSM after partition is shown in Figure 2-4. The total number of feedback marks is reduced from 6 to 2. Two iteration loops exist namely within tasks B, D, G and within tasks A and C.

	F	B	D	G	C	A	E
F							
B				X			
D		X					
G	X		X				
C	X	X		X		X	
A			X		X		
E	X				X		

Figure 2-4 A DSM after partition [21]

Tearing is the process of choosing the set of feedback marks that if removed from the matrix (and then the matrix is re-partitioned) will render the matrix lower triangular. The marks that we remove from the matrix are called "tears" [21]. This is a process in which familiarity of the problem and profound domain knowledge would be helpful. It is because tearing is not simply a mathematical operation on a matrix. The tears bear with them practical constraints when we are talking about a real project. Being able to identify the correct tears can improve the productivity of a process hence greatly reduce total project execution time [9].

Extensive research has been done in partition and tearing algorithms. Details can be found on the DSM website [21].

2.2.4 Resemblance to Linear Systems

The main structure of a DSM is a square matrix. Naturally we should expect it to resemble a linear system, with which many existing mathematical treatments can be applied. One of them is the eigen-structure analysis proposed by Robert P. Smith [21]. Below is a brief description of this approach.

The DSMs can be analyzed by analogy to dynamic linear systems. In study of such systems, one thing that we are always interested in is the eigenvalues. Each eigenvalue tells us about the convergence rate of a dynamic mode. We need to pay attention to modes that have slow convergence rates, since these will dominate the iterative behavior and time taken. The slow convergence rate modes are those that have large magnitude eigenvalues.

Since eigenvectors are associated with eigenvalues, the eigenvectors associated with the large-magnitude eigenvalues are of particular interest. The entries that are heavily weighted in these eigenvectors describe those tasks that will be the primary components of the slowly converging iterations. From a management perspective these are interesting since they tell us about which are the most important of the iterative, coupled tasks. They are the tasks that will have to be reworked a significant number of times. If we can reduce the rework probability, or rework impact of these tasks, by means of additional resource or better technology, the total project execution time can be reduced significantly, too.

To estimate the total time needed by a project, we can take the following approach. Let $\boldsymbol{u_t}$ be a vector of remaining work on each task at iteration t, and let $\boldsymbol{M}$ be the product (element-by-element multiplication) of the two planes of the square numerical DSM and a rank-one matrix whose columns are the learning curve effect vector. Since a DSM shows the relationship between task rework, we can assume that work completed in the $(t+1)$th iteration is a linear function of work completed in the (t)th iteration, with the linear weights being the numerical values in the matrix $\boldsymbol{M}$. Therefore work in the $(t+1)$th iteration is $\boldsymbol{Mu_t}$.

At time zero the initial work remaining on all tasks is 1. (There is 100% of work remaining for every task.) Therefore $\boldsymbol{u_0}$ is a vector of all 1s. $\boldsymbol{u_0}$ is known as the initial work vector. After the first iteration the work remaining is $\boldsymbol{Mu_0}$. After the second iteration the work remaining is $\boldsymbol{M^2u_0}$. After the (n)th iteration the work remaining is $\boldsymbol{M^nu_0}$. Each of these terms is known as a work vector.

If we sum up all of the work vectors we obtain the total work completed in the span of the project. This sum is similar to the sum of a *geometric progression* (GP), except that the common factor is the matrix $\boldsymbol{M}$ instead of a constant. Therefore the sum to infinity can be calculated directly using GP formula $\boldsymbol{U = (I - M)^{-1}u_0}$.

In addition, let $\boldsymbol{W}$ be a diagonal matrix which contains the task durations along its diagonal, then $\boldsymbol{T = WU}$ is a vector which contains the amount of time that each task will require during the (n) iteration stages. Finally, we sum up all the components of $\boldsymbol{T}$ to get the total project duration.

A limitation of this approach is that it only works for DSMs whose derived matrix $\boldsymbol{M}$ has eigenvalues smaller than 1. This is in analogy to the requirement that the common factor of a GP must be less than 1 in order for the sum to infinity to exist. Otherwise the ill-conditioned matrix will cause problem in the matrix-inversion step. When a DSM is found to be ill-conditioned, we have to check whether it is constructed correctly. If the data captured in the DSM is valid, it means the project itself is unstable and will never finish. In that case, we have to revise the feasibility of the project.

2.3 Chapter Summary

In this chapter, basic features of DSMs that will be used in our simulation-based optimization model are introduced. The advantages of using DSM as a representation and modeling tool are also discussed. These advantages will become more obvious as we go into the simulation and optimization aspects of our work.

CHAPTER 3
DSM-BASED MONTE CARLO SIMULATION

Monte Carlo simulations is a class of simulations that rely on repeated random sampling to compute results. They are commonly used when it is infeasible or impossible to compute an exact result with a deterministic algorithm.[25] This chapter presents a DSM-based Monte Carlo simulation of a product development project. It is an extension of Browning's work on DSM simulation [3][4].

The goal of the simulation is to derive estimates of product development project duration based on the constituent tasks and their relationships. To understand the amounts of uncertainty and risk present in such estimates, one can examine a distribution of schedule outcomes. The model randomly samples activity characteristics from their distributions and combines them to simulate a process result. Doing this many times yields distributions of outcomes and enables analysis of these distributions for the amount of risk they imply.

In our simulation model, we modify some of the work policies proposed by Browning. The reasons will be discussed in detail in the subsections that follow. In addition to the precedence relations and iterations that are modeled by Browning, we have imposed resource constraints to the projects. The relationship between resource levels and task durations, as well as the relationship between resource levels and rework, which are not addressed in earlier literatures, are also included in our new simulation model.

3.1 Simulation Inputs

Table 3-1 shows the input data that are needed for the simulation. Items 1 through 5 have already been introduced in section 2.2.2. They are all attributes of a numerical DSM. Items 6 through 10 are resource related attributes which are newly included in the simulation. We shall describe each of them in detail.

In Browning's model, the tasks only need to satisfy the precedence requirements in order to get started. In our model, apart from the precedence relations, a minimal resource requirement also needs to be satisfied.

To define these resource constraints, we first specify a set of renewable resources that are needed to execute individual tasks of a project. These resources are assumed to be available per period in constant amounts. Item 6, resource available, is a constant row vector that captures this piece of information. Each component of the vector represents the available quantity of one particular type of resource.

Item 7, resource requirements, are the nominal values of the amounts of resources required by the tasks. These values can be positive or zero. They are stored as a matrix. Each column of the matrix represents one type of resource, whereas each row represents resource requirements for one task. As an example, an entry in row 3 column 2 represents the amount of resource of type 2 that is needed by task 3.

	Input Data	**Remarks**
1	List of tasks of the project	Number of activities
2	Duration estimates	Include BCV, MLV and WCV, in units of time, days, weeks etc
3	Learning curve effects	Percentage of the original value of the intended work when a task is reworked, range [0,1]
4	Rework probabilities	Likelihood of a task being reworked when the input information changes, range [0,1]
5	Rework impacts	Percentage of activity to be reworked [0,1]
6	Resources available	Amount of resource of each type at each time period
7	Resource requirements	Amount of resource of each type needed by each task
8	Resource requirement variation bounds	Lower and upper bounds for resource requirement variation, expressed as percentages of the nominal values
9	Rework probability variation bounds	Lower and upper bounds for rework probability variation, expressed as percentages of the nominal values
10	Task duration variation bounds	Lower and upper bounds for task duration variation, expressed as percentages of the original task durations
11	Run size	Maximum number of simulation runs

Table 3-1 Simulation inputs

Even though we assume that most of the time, tasks are executed with the amounts of resources equal to their nominal values, we also allow small variations when there are abundant resources or when resources are scarce. This means tasks may be executed with extra amounts of resources or reduced amounts depending on the availability. It is accounted for by resource requirement variation factors, which is the ratio of the actual amount of resources allocated to the nominal value of the resource requirement. These variations are limited to the range defined by Item 8, the upper and lower bounds. The bounds are expressed as percentages of the nominal values. By allowing resource requirement variation, we add flexibility to our model.

On one hand, when the resources are scarce, tasks are still allowed to start as long as the minimal resource requirements are met. However, it comes with a penalty. A reduced amount of resource may give rise to some compromise in work quality. It is likely that the output becomes less reliable. As a result the task itself, as well as its dependent tasks, have higher probabilities to be reworked. Another possibility of working with inadequate resource is that a task may need a longer duration in order to be completed.

On the other hand, when the resources are abundant, we can allocate more resources to the tasks. The advantage of working with more resources is probably more reliable, and better quality work. In this case, we may assume that the probabilities of rework decrease. Correspondingly, we may also expect that the additional resources can be used to shorten the task durations. However, in either case there is a decreasing margin of return. We would expect the amount of benefit to saturate or even deteriorate when the amounts of resources allocated to a particular task reach certain levels. Therefore setting an upper bound for the resource requirement prevents waste as well as adverse effects associated with over-allocation. Item 9, rework probability variation bounds, and Item 10, task duration variation bounds, are two attributes that are in close relation with Item 8, resource requirement variation bounds.

In our model, we assume two mutually exclusive working modes in which a task can be executed whenever the amounts of resources allocated deviate from their nominal values: either the task duration is varied to match with the variation in resource levels while keeping rework probabilities fixed, or the rework probabilities are varied while the task duration remains unchanged.

Moreover, we assume that the rework probability and the amounts of resources allocated have a piecewise linear relation within the lower and upper bounds. Similarly, we assume a piecewise linear relation for task duration and the amounts of resources allocated. To illustrate this point, we look at the following example.

Assume we have the following data for a task of a project.

	Resource Type A	Resource Type B	Resource Type C
Resource Requirements	10 units	20 units	30 units
Resource Requirement Variation Bounds	[−0.1, +0.1]		
Rework Probability Variation Bounds	[−0.1, +0.2]		
Task Duration Variation Bounds	[−0.1, +0.1]		

Table 3-2 A fictitious example of resource related data for a task

Figure 3-1 illustrates the relationship between the actual amounts of resources allocated to the task and the associated rework probabilities. Since the rework probability bounds are not symmetrical about the nominal value, the rework probability-resource level variation chart is also asymmetrical.

Figure 3-2 shows the relationship between actual amounts of resources allocated to the task and the associated task durations. The task duration-resource level variation chart is symmetrical since the bounds are symmetrical about the nominal values.

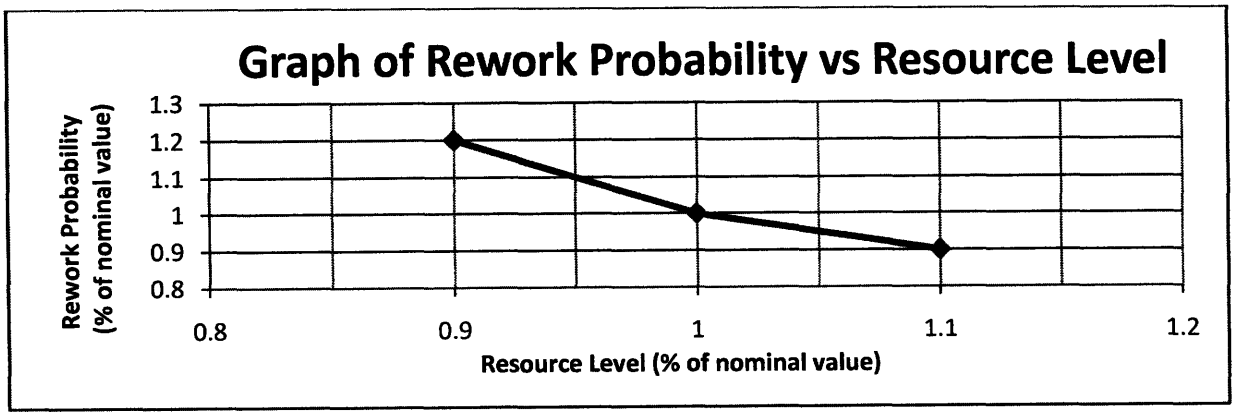


Figure 3-1 Rework Probability - Resource Level variation chart

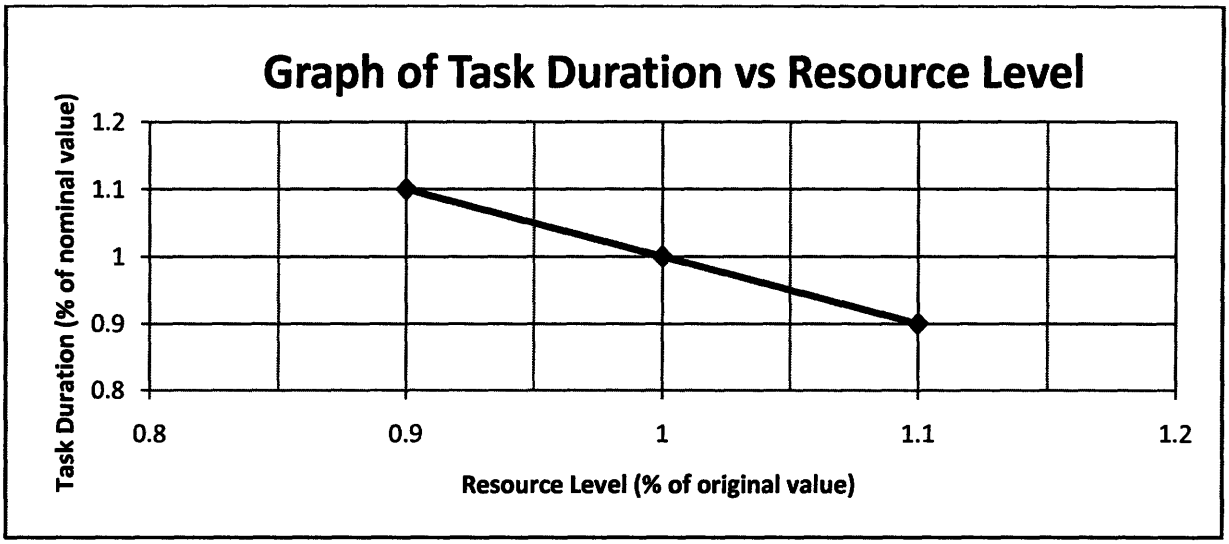


Figure 3-2 Task Duration – Resource Level variation chart

	Actual Amounts of Resources Allocated			New Rework Probability (% of nominal value)	New Task Duration (% of original value)
	Resource Type A	Resource Type B	Resource Type C		
1	10.5	21	31.5	0.95	0.95
2	10	20	30	1	1
3	9.5	19	28.5	1.1	1.05

Table 3-3 Pairs of resource allocation schemes and rework probabilities

In Table 3-3, three different resource allocation schemes result in three different modifications to the rework probabilities and task durations. Higher levels of resources allocated correspond to smaller rework probabilities or shorter task durations and vice versa.

Two assumptions are made here. Firstly the total amounts of resources available are more than any single task's minimal resource requirements. That means, at any time, at least one task can be executed. Secondly, we never unnecessarily allocate extra resources. In the above example, the scenario of allocating 10 units of resource A, 23 units of resource B, and 30 units of resource C will never happen. The additional 3 units of resource B (as compared to case 1 in Table 3-3) is assumed to have no influence on the rework probability or task duration, hence should be removed.

3.2 Platform

The old DSM simulation algorithm is implemented using Excel macros. However due the limitation of memory size, the old routines can only be used to deal with relatively small DSMs. MATLAB is a software package that analyzes and visualizes data by using existing functions and user-designed programs. Apart from larger memory and better computing powers, it also has convenient and efficient matrix manipulations, which are added advantages to the DSM-based simulation optimization model. It is chosen as the primary platform for this thesis. The numerical DSMs, and most other related variables are conveniently stored in MATLAB as matrices and vectors.

3.3 Simulation Algorithm

To facilitate the presentation of the algorithm, we define the symbols used as listed in Table 3-4.

Symbol	Description
n	number of tasks
m	number of types of resources
$TaskDur$	n×1 vector of initial task durations
$WorkRemaining$	n×1 vector of the amount of unfinished work for the tasks
$MayWorkNow$	n×1 binary vector representing potential concurrent tasks based on precedence relationships alone
$WorkNow$	n×1 binary vector representing the tasks that are worked on at the current stage
$WorkThen$	n×1 binary vector representing the tasks that are worked on at the previous stage
RA	1×m vector of available resources
Res	n×m matrix, resource requirements of the tasks for each type
$Ractual$	n×m matrix of actual resource allocation
$TimeStep$	time advancement, corresponding to the shortest task duration at each stage

Table 3-4 List of symbols used in the simulation algorithm

The detailed DSM simulation algorithm steps are described and explained bellow.

Step 1. Sample task durations from their triangular probability distributions, using BCVs, MLVs, and WCVs.

We are interested in modeling stochastic task durations. Instead of using fixed values for all the task durations, we sample each of them from a triangular distribution at the start of every simulation run. A triangular distribution is a simple distribution that is characterized by the three values BCV, MLV and WCV. Figure 3-3 shows the probability density function and the cumulative distribution function of such a distribution. In the figure a corresponds to BCV, c corresponds to MLV and b corresponds to WCV.

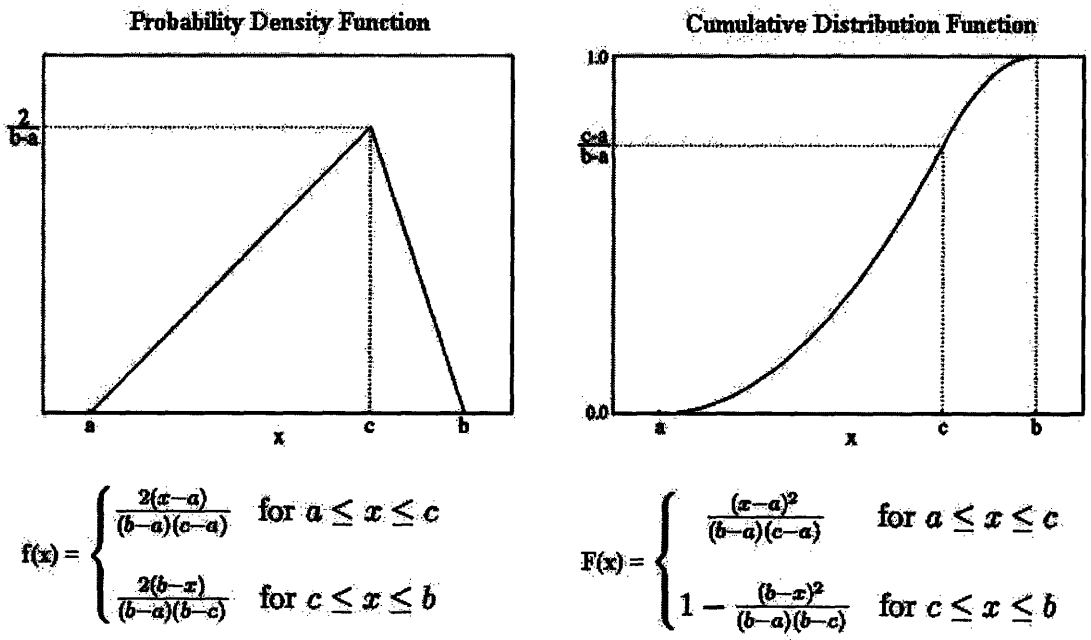


Figure 3-3 PDF and CDF of a triangular distribution [26]

Step 2. Initialize $WorkRemaining$ vector: $WorkRemaining = TaskDur.$

At the beginning of a project, the amount of work remaining for a task is the same as the duration of that task. In subsequent stages as time elapses, the amount of work remaining becomes smaller and smaller.

Step 3. If *WorkRemaining* is equal to the zero vector, all tasks have been finished. One simulation run is ended. Stop the iteration. Otherwise continue with Step 4.

Step 4. Reset *MayWorkNow* and *WorkNow* to zero vectors, *Ractual* to zero matrix, and *RA* to its original values.

Step 5. Check the precedence relationships to find the set of tasks that can be started concurrently. Set the corresponding components of *MayWorkNow* to 1.

To identify the set of potential concurrent tasks, we loop through the columns of the DSM corresponding to tasks with unfinished work. If a task does not depend on other tasks for input (an empty column), it can be executed in the current stage. If a task needs input from upstream tasks (presence of marks above the task in its column), and if those upstream tasks have been finished, the task may be started in the current stage as well.

In Browning's work, he only allows tasks within the same band to work concurrently. That means only consecutive tasks are allowed to work concurrently. We adopt a less stringent rule: as long as the two criteria mentioned previously are met, a task can potentially work with the others, subject to additional resource constraints. In reality, if all the information inputs of a task are ready, as long as the necessary resources are available, we would expect the project team to start working on the task immediately instead of waiting for a later time and leaving the resources idle. Therefore removing the bandings may make the model more realistic. In addition to that, adopting such a rule allows easy extension of the model to represent multiple projects. To switch between these two work policies, only small modification to the MATLAB code is needed.

Moreover, the previous simulation model does not allow pre-emption: once a task starts, it runs to completion. The non-pre-emption rule has its advantages. Continuity is one of them. We would expect constant interruption of work to make workers less productive. Moreover, causing certain individuals or groups to stop work temporarily requires clear management policies that are not present in many engineering environments.[3] In our model we allow pre-emption, but with a penalty. A previously worked task may be interrupted and delayed, but a certain

percentage of work will be added to its remaining work. This penalty represents the additional time needed to pick up a half-finished task.

Step 6. Check the resource constraints. If the total amount of resources required is less than or equal to the available amount, set $WorkNow = MayWorkNow$ and continue with Step 8. Otherwise a conflict arises, continue with Step 7.

Step 7. Rank all the tasks that can be started at the current stage if the resource constraints were not there.

This is the most crucial part of the scheduling process, since the rankings directly determine which tasks should receive the necessary resources and get started at the current stage. There are many heuristic rules that have been used in existing literatures to rank the tasks. Most commonly used heuristic rules include *Shortest Operation First* (SOF), *Longest Operation First* (LOF), *Minimum Slack First* (MSF) etc. In other cases, tasks are simply ranked according to user preferences, where the preferences are usually derived from accumulated experience and/or other real constraints faced by the project.

It is difficult to schedule a complex project merely based on experience, since what experience can tell us is limited after all. Projects differ from one another; experience gained from one project may not be applicable to others. Experience is not something that is easily available or applicable, therefore we need some general yet well-defined rules which can be easily applied when we have a new project at hand. That is why heuristic rules come in handy.

If the project structure is simple and monotonic, a single heuristic rule may be sufficient to generate an optimal scheduling scheme. However, when the structure is complex, sticking with a single heuristic rule throughout the project is definitely too simplistic an approach. Just as we see that certain heuristic rules work better for certain types of projects but not all, we would naturally expect that at different stages of a project (which we may consider as sub-projects) the rules used for ranking should also be adjusted to match with the characteristics of the project at those stages.

In this thesis, ranking is based on a weighted sum of several chosen ranking factors. Each factor can be thought of as a single heuristic rule. Assigning different weights to these factors is approximately equivalent to applying the heuristic rules in different orders and different proportions. A task that scores low for a heavily weighted factor may be ranked low overall. On the other hand, a task that scores extremely high for a less heavily weighted factor may receive a high overall ranking. Therefore by wisely manipulating the weights, we can almost obtain any kind of task rankings that is desired at different stages. If we are able to tell which factors play more important roles in affecting project duration at a particular stage, we can then assign heavier weights to those factors. The corresponding rankings will result in task sequence that shortens the total project duration. We shall introduce the ranking factors below, but leave the discussion about how weights are chosen to the subsequent chapter about DSM optimization.

After examining all the project related inputs, we have chosen four ranking factors, which we deem to play the most important roles in affecting the total project duration. The four ranking factors are namely the critical time, the amount of possible rework, resource scarcity and possible penalty for pre-emption. We have also experimented with other factors, such as the amount of work remaining for a task as a percentage of the total remaining work, the number of dependent tasks etc. They either do not affect the total project duration much or, they can be accounted for by the four factors we have chosen.

The first factor, critical time, comes from the *Critical Path Method* (CPM). It is the total time from the start of a task to the end of the critical path or sub-critical path that the task is involved in. In a sense, it reflects the amount of work that can only be finished after a particular task starts. A task that is critical should be started as early as possible, so that the project is less likely to be delayed due to delays of the tasks on the critical path.

The critical path and sub-critical paths of a project can be found using the backflow algorithm [22]. It is implemented using data presented in a DSM as follows. Extract the task durations from the main diagonal of the DSM. Convert the 2-plane numerical DSM to a two-dimensional binary DSM, and remove all the super-diagonal marks (hence all the rework is ignored in this part of the analysis). Look for empty columns in the DSM, they corresponds to the end tasks of the paths. The critical time of these tasks are their own task durations. Temporarily remove these tasks and

their associated dependency relations from the DSM, and continue to look for empty columns. They corresponds to the 2nd last tasks of the paths. To find their critical time, first look at their corresponding columns in the original lower-triangular DSM to find their successors. The critical time of these tasks is the sum of their own task duration and the maximum successor duration. Continue with the process until the critical time of all the tasks have been computed. The maximum value of all the critical times is the critical path length.

The second ranking factor has two components, namely the first-order rework and the second-order rework. First order rework refers to rework generated due to information feedback from a downstream task to upstream dependent tasks. Second order rework is the rework on downstream dependent tasks, when an upstream task is repeated. To find the first-order rework that can be possibly generated by a task, we look for super-diagonal entries in the column of that task. To find the second-order rework, we first locate the columns that correspond to tasks with potential first-order rework. In these columns, all the sub-diagonal entries are sources of second-order rework. The rework is computed as the product of the rework probability, rework impact and learning curve effect. We sum up all these possible rework to get the second ranking factor.

The third ranking factor involves resource considerations. We define the scarcity factor of resource type k as follows:

$$scarcity\_factor_k = \frac{\sum_{i \epsilon U} Res(i,k)}{RA(k)} \quad for\ k = 1,2 \ldots m,$$

where U is the set of unfinished tasks, $Res(i,k)$ is the amount of resource of type k required by task i, and $RA(k)$ is the available amount of resource of type k.

The normalized scarcity factor is:

$$norm\_scarcity\_factor_k = \frac{scarcity\_factor_k}{\max\limits_{j \in [1,2,\ldots,m]} (scarcity\_factor_j)}.$$

The third ranking factor is calculated as:

$$factor_3 = \left[\max_{k\in[1,2,\ldots,m]}(Res(i,k) \times norm\_scarcity\_factor_k)\right] \times WorkRemaining(i)$$

$$for\ i \in U$$

If the resource requirements of a task are low, this factor does not tell us much about whether it should be delayed or not, since it will only contribute a small fraction to the overall ranking. In this case, other factors will play a more important role in determining the ranking of this task. However, if a long-duration task requires a large quantity of a scarce resource, it will score very high for factor 3. As a result, its overall ranking may be high as well, which means this task may be started early. This has some advantages. Firstly, it is highly unlikely that we can avoid resource conflict even if we delay this task to a later stage. Moreover, if we can finish it early, the scarce resource can be released early, which makes later stages much easier to be managed. Of course, the overall ranking still depends on the weights assigned to the factors, which is left to be determined by the optimization algorithm.

The last ranking factor is the penalty due to interruption of a started task. Even though disruption of an unfinished task is discouraged under normal circumstances, it may be beneficial to delay a task occasionally if certain scarce resources can be released for better use. If the penalty of delaying a task is not large, the task may be delayed if the delay is deemed more beneficial. On the other hand, if the penalty is high, the task is likely to be ranked highly and will not be delayed. This factor only applies to unfinished tasks that have been worked on in the previous stage. One exception is that if an unfinished task is interrupted due to rework of its predecessors, we consider it as a "strategic wait" [3], hence no penalty is added. In other words, it only assumes a non-zero value for task (i) when the expression

$$WorkThen(i) == 1\ \&\&\ MayWorkNow == 1\ \&\&$$

$$WorkNow(i)\sim = 1\ \&\&\ WorkRemaining(i)\sim = 0$$

is evaluated to be true.

The highly-ranked tasks have higher priorities to be assigned the necessary resources to get started. The original DSM task order acts as a tie breaker.

Step 8. Allocate the necessary resources to the tasks in the original DSM task order (if no resource conflict presents) or in the order of rankings (in the case of a resource conflict). Update $Ractual$, RA and $WorkNow$ accordingly.

Always try to assign the resources according to the nominal values to all the tasks first. If the amount of resources available is not enough to cover the nominal resource requirement levels of a task, try the highest possible value that is above the lower bound. Otherwise skip the current task. If after one round of resource allocation there are still idling resources, assign additional resources to the top-ranking tasks, observing the upper bounds of resource requirement variation.

Step 9. If the actual amounts of resources allocated is different from the nominal values of the resource requirements, decide on the working mode, and update either the rework probabilities or the task durations accordingly.

As mentioned earlier, whenever the actual resource levels deviates from the nominal values, one can choose whether to modify the rework probabilities or the task durations, but not both. Again, the most suitable working mode of each task is chosen by the optimization algorithm for individual projects.

Step 10. Find the time advancement, which corresponds to the smallest amount of work remaining for the active set at the current stage.

Step 11. Update project execution time and the $WorkRemaining$ vector. For all the newly finished tasks generate first order and second order rework for their dependent tasks.

Generation of rework is similar to the method used to calculated the second ranking factor. The only difference is that instead of multiplying the rework probability to the product of rework impact and learning curve effect, it is compared with a random number in the range of 0 to 1. If

the random number is smaller than the rework probability, rework is generated and added to the $WorkRemaining$ vector. Otherwise, no rework is generated.

If the work remaining for any particular task is greater than the original task duration, adjust it to be 90% of the original duration to represent some learning. Continue with Step 3.

The simulation is normally ran many runs until the mean and standard deviation of the total project duration stabilize. By plotting the histogram, the general distribution of the total project duration can be obtained.

3.4 Chapter Summary

In this chapter, we present a detailed description of a DSM-based simulation algorithm. The simulation accounts for stochastic task durations, task concurrency, and stochastic rework, while observing precedence rules and satisfying resource constraints. The differences from previous works and the reasons for these modifications are emphasized.

CHAPTER 4
OPTIMIZATION USING GENETIC ALGORITHM

We have already discussed the DSM-based representation, modeling and simulation of a product development project in earlier chapters. The ultimate goal is for us to gain insights into the managerial aspects hence be able to schedule the project tasks efficiently. This includes being able to complete the project as early as possible, reduce the amount of uncertainties and risks, as well as to use the limited resources wisely. In this chapter we first give an overview of the literatures on *Resource-Constrained Project Scheduling Problems* (RCPSP). Then our optimization problem is formally defined. Last but not least, we introduce the optimization method, genetic algorithm, that we use to solve our optimization problem. Both the general approach and problem specific aspects will be discussed.

4.1 Background of RCPSPs

The development of large-scale projects after WWII has led to a plethora of research in project scheduling. The initial research focused on determining the start times for tasks so as to minimize the overall project duration while satisfying the precedence requirements. The *Program Evaluation and Review Technique* (PERT) and the *Critical Path Method* (CPM) are the earlier models used for analyzing scheduling problems [3]. CPM is useful when the task durations are deterministic. PERT can be used for projects with stochastic durations. However, neither of them can handle resource constraints and feedback relationships between tasks.

An extension of PERT called the *Generalized Evaluation and Review Technique* (GERT) was developed later. It enables simulation-based analysis which accounts for iteration. However it does not allow dynamic activity concurrency or variable rework impacts. [3]

Several branch-and-bound algorithms have been developed to solve scheduling problems with resource constraints.[10][11] Researchers have formulated efficient bounds that cut down the

computation while traversing a branch or backtracking. These algorithms assume acyclic networks, which does not account for iterations.

Many heuristic-based approaches can also be found in literature. Most frequently used are priority rules such as *first come first serve* (FCFS), *shortest operation first* (SOF), *minimum slack* (MINSLK), etc. These rules are applied to different RCPSPs according to the different characteristics of the individual problems. Design of experiments approach is used to study how well each priority rule works for a specific type of project [6]. However, most of these heuristic-based approaches require experience and sound judgement at critical stages of the project, which is difficult. It is also not guaranteed that a single rule will work optimally when used for the whole project span.

Genetic algorithms used for DSM-based optimization problems can be found in existing literatures [19][20][24]. Some of these works do not consider iterations at all [19][24]. Others do not deal with iterations explicitly [20]. The amount of possible rework is estimated based on the rework probabilities, and rework impacts and added to the original task durations. In a sense, the problem is transformed to a RCPSP without iterations.

In this paper, genetic algorithm is used to optimize a set of heuristic weights, which defines a ranking scheme used whenever resource conflicts occurs in a simulation. The heuristics used combines information specifically derived from the project structure, and takes into account possible effect of the current decision on future events.

4.2 Optimization Problem Formulation

A general formulation of an RCPSP can be expressed as follows [10]:

$$\min_{starting\ times} f_n$$

$$subject\ to \qquad (1)\ f_j - f_i \geq d_j, \quad (i,j) \in H$$

$$(2) \sum_{S_t} r_{ik} \leq b_k, \qquad t = 1,2,\dots,f_n, k = 1,2,\dots,K, where$$

f_i = finish time of task $i, i = 1,2, \ldots n$, and task n represents the last task

H = set of pairs of tasks indicating precedence constraints

d_i = duration of task i

r_{ik} = amount of resources of type k required by task i

S_t = set of tasks in process in time interval $(t-1, t] = \{i | f_i - d_i < t \leq f_i\}$

b_k = total availability of resource type k.

Problems with such a formulation typically considers projects that can be represented by acyclic activity-on-node digraphs. (1) represents the set of precedence constraints. (2) represents the resource constraints. Using our DSM representation, the following correspondence can be established.

Symbols Used in the General Formulation of RCPSP	DSM Representation
H	sub-diagonal entries
d_i	diagonal entries of the 1st plane of DSM
r_{ik}	entry in row i column k of matrix Res
S_t	vector $WorkNow$
b_k	kth entry of vector RA

Table 4-1 Symbol comparison

We can see that our DSM simulation already satisfies all the constraints in this general formulation of the RCPSP. Therefore, we adopt a similar formulation in our single-objective optimization. However, since the task durations are stochastic, and rework is also generated randomly, it is impossible for us to find a fixed value for the finishing times of the tasks. Therefore we modify the objective function to be the mean value of the total project duration. The objective value is obtained by running the DSM-based simulation a fixed number (a large number so that we can observe a stable distribution) of times.

In the general formulation, the decision variables are the starting times of the tasks (assuming non-pre-emption rule). Once they are fixed, a scheduling solution is fixed. In our problem due to the presence of stochastic task durations and rework, the starting times are no longer fixed

quantities for each simulation. Therefore we can only specify a rule with which we order the tasks. As we can see from the discussions in chapter 3, such a rule is defined by the weights assigned to the ranking factors. Given a project and all its relevant data for DSM simulation, the four ranking factors can be directly computed from these data. If we were to fix the weights on these ranking factors, we would have implicitly defined a scheduling solution. In addition to fixing a rule for ordering the tasks, we also specify the work mode for each task. Therefore in our formulation the decision variables become the weights on the ranking factors and the working modes of the tasks, instead of the starting times of the tasks.

Our optimization problem is stated as follow:

$$\min_{weights,work\ modes} J = mean\ total\ project\ duration$$

$$subject\ to \quad precedence\ and\ resource\ constraints$$

4.3 Genetic Algorithm

Genetic algorithms (GA) are statistical methods used to solve optimization problems, among other applications. They mimic the evolution of living things. "Survival of the fittest" is the underlying principle.

A GA normally starts with an initial population representing possible solutions randomly picked from the search space of a particular problem. The fitness of each individual is evaluated based on objective functions derived from the problem at hand. The fittest individuals will then be selected and allowed to reproduce by means of crossover. This is followed by mutation, a step that injects diversity to the population. Finally, the old population is replaced by the new population which consists of the parents and their offspring. This process is repeated until a certain number of generations have elapsed or the mean fitness of the population has been stabilized for a predefined number of generations.

The GA used in this project is based on the Little Genetic Algorithm as described in [8], with modifications on population representation and crossover operation. It employs a basic form of genetic algorithm with a fixed-size population, Gray binary coding, fitness-proportional or

Roulette wheel selection, and bit-wise mutation. Whether a fixed number of generations have elapsed is chosen as the stopping criterion.

In the following sub-sections, we first describe the coding, selection, crossover and mutation operators which are more or less standard in most genetic algorithms, followed by details pertaining particularly to this project.

4.3.1 Initial Population and Coding

Rather than starting with a single solution in the search space, GA starts with a group of solutions (or rather guesses of the optimal solution) located randomly within the search space. This is termed the initial population, or generation 1.

Initially, we define our population to be consisting of a fixed number of ranking schemes, each represented by a set of four weights, coded into Gray binary strings. We would like the weights to have a sensitivity of 0.01. Therefore each weight is represented by a 7-bit substring. The four substrings are joined together in sequence to form a single string, as shown in Figure 4-1.

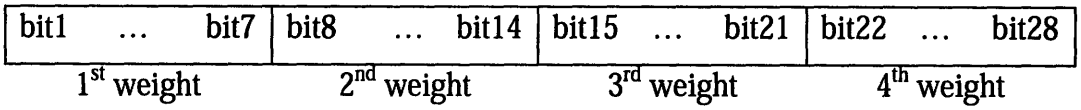


Figure 4-1 String representation of an individual

Gray binary strings, like the normal binary strings, are made up of only 0s and 1s. The difference is that adjacent integers in Gray strings differ only by one bit position. Traversing through an integer sequence represented by Gray strings therefore involves flipping only one bit each step. For example for a normal binary representation of the integers from 0 to 3 {(00), (01), (10), (11)}, in order to change from 2 to 3, we need to flip both bits; but for Gray strings {(00), (01), (11), (10)}, we only need to flip the most significant bit from 0 to 1 for the same change from 2 to 3. Table 4-2 compares the normal binary strings with the Gray strings for a string length of 4.

Gray coding preserves the correspondence between the genotype metric (bit-to-bit difference or Hamming distance) and the phenotype metric. This property of Gray strings has significant implications in mutation. "Most mutations will make only small changes, while the occasional mutation that effects a truly big change (for example by flipping the LSB of '000', we get '001', a rather significant change from 0 to 7) may initiate exploration of an entirely new region in the search space." [13]

Integer	Binary	Gray		Integer	Binary	Gray
0	0000	0000		8	1000	1100
1	0001	0001		9	1001	1101
2	0010	0011		10	1010	1111
3	0011	0010		11	1011	1110
4	0100	0110		12	1100	1010
5	0101	0111		13	1101	1011
6	0110	0101		14	1110	1001
7	0111	0100		15	1111	1000

Table 4-2 Comparison between binary and Gray strings

The algorithm to convert a Gray string to a normal binary string is given below [14]:

$$B(1) = G(1);$$

$$for\ i = 2\ to\ N$$

$$B(i) = B(i-1)\ xor\ G(i);$$

$$end$$

To convert a normal binary string to a Gray string we can use the following algorithm [14]:

$$G(1) = B(1);$$

$$for\ i = 2\ to\ N$$

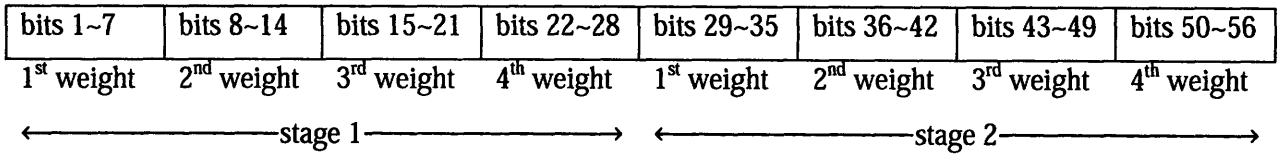
$$G(i) = B(i-1)\ xor\ B(i);$$

$$end$$

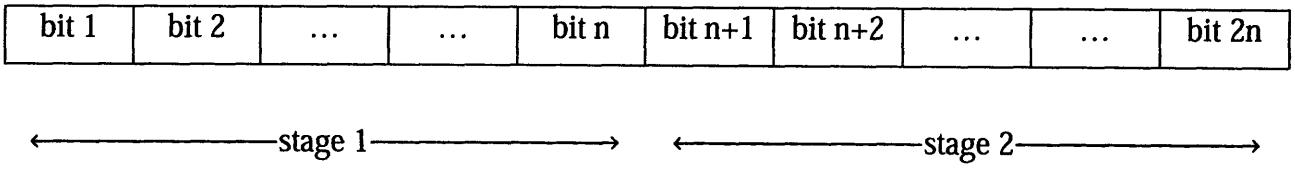
Here, $B(1)$ and $G(1)$ both represent the most significant bits.

In our simulation model, we introduce two work modes, used whenever the amounts of resources allocated deviate from the nominal values of the resource requirements: either the task duration is adjusted (call it the T mode) or the rework probabilities are adjusted (call it the P mode). Therefore, apart from the Gray string that is used to represent the weights of each scheduling solution, we use another two separate binary strings, each of length n (which is the number of tasks), to represent the preferred work modes for each task when resource level is low and when resource level is high, respectively. A zero means T mode is preferred, whereas a one means P mode is preferred.

After some initial testing of the algorithm, however, we decide to improve the GA solution by dividing the project into stages, and assign separate set of weights and work modes to each individual stages. In this case, a 2-stage project will have strings of lengths twice as long as those in the original case, as shown in Figure 4-2. When the projects are big, a large number of tasks will be involved. We expect the project structure to vary greatly from the earlier stages to the later stages. If the whole project is forced to be scheduled based on one set of weights, very likely the weights obtained by GA will only achieve mediocre performance since the distinct features of the different stages tend to average out when we look at the whole span of the project. Just like we avoid using a single heuristic rule for the whole project, we naturally avoid using a single set of weights for the whole project span.



(a) Gray string for the weights



(b) Binary string for the work modes
(same structure for both low and high resource levels)

Figure 4-2 Modified string representation of an individual

4.3.2 Fitness Evaluation

To evaluate the fitness values, we need to first decode the strings representing each individual. After we obtain the weights and work modes for the different stages, we can then run the simulations according to the task sequence and work modes suggested by the individual solutions.

For each weighting scheme, we allow the simulation to run a fixed number of runs. Task durations are sampled from triangular distributions and rework are generated stochastically each time. At the end of these runs, we calculate the mean and the standard deviation of the total project duration. The mean measures the quality of the scheduling solution that is defined by the ranking scheme. The standard deviation measures the uncertainties or risks involved by adopting the scheduling solution. Both parameters are important from a management point of view and should be minimized.

The objective function of our optimization problem is defined as

$$\min_{\textit{weighting schemes,work modes}} \textit{mean total project duration.}$$

The fitness of an individual is the negative mean total project duration obtained from the Monte Carlo simulation.

4.3.3 Selection

In order to derive a new generation, the old population needs to undergo a selection process in which only the better performing individuals will be allowed to stay on or reproduce.

A commonly used selection method is the *fitness-proportional* or *Roulette wheel* selection. The basic idea is that an individual with a higher fitness value will have a higher chance to be selected. To deal with the negative fitness values, we shift all of them by the same amount such that the lowest fitness value becomes zero. Let the sum of the new fitness values of all individuals be the circumference of the Roulette wheel. Then an individual with a higher fitness

value will occupy a bigger segment of the wheel. When the wheel is rolled and a ball thrown in, the chances that the ball falls into a bigger segment will be greater than it falling into a smaller segment. Mathematically, the ball is simply represented by a random number in between 0 and the sum of the fitness values. The fitness values of the individuals are added one at a time until it just exceeds the random number. The last individual to be added will be selected. [8] The selected individuals may walk into the next generation with or without crossover. It is up to the programmer to decide on the crossover probability.

4.3.4 Crossover

Crossover is analogous to the sexual reproduction in nature. New individuals are produced with the exchange of genetic information between the two mating individuals. This allows different traits possessed by the parents to be recombined in the hope to generate children with better overall characteristics.

Prior to crossover, the weights are first normalized such that the four weights of a single stage sum up to 1. The normalized weights are re-coded into Gray strings for crossover. This step is necessary for preserving continuity from one generation to the next. We know that only the relative magnitudes of a set of weights affect the rankings, whereas the absolute values do not matter. The set of weights (0.1, 0.1, 0.1, 0.1) is effectively the same as another set of weights say (0.8, 0.8, 0.8, 0.8), meaning the four ranking factors are equally important. If the weights are not normalized before crossover, we may end up with weights like (0.1, 0.1, 0.8, 0.8), and (0.8, 0.8, 0.1, 0.1) which tell us that some factors are much more important than the others. In this case, the crossover process becomes a totally random process, since almost all the traits of the parents are lost. Therefore normalization is used to prevent such detrimental effects.

Single-point crossover is carried out within individual stages. Consider two mating strings from a two-stage project:

$S_1 = (B_{11}(1) \dots B_{11}(N)\boldsymbol{B_{12}(1)} \dots \boldsymbol{B_{12}(N)})$ and $S_2 = (\mathbb{B}_{21}(\mathbb{1}) \dots \mathbb{B}_{21}(\mathbb{N})\mathrm{B}_{22}(1) \dots \mathrm{B}_{22}(\mathrm{N}))$,

where N is the number of bits in the substrings of each stage. We need to select two crossover sites in this case. Randomly select a bit in the first substring, say the kth bit, such that $1 \leq k < N$. Randomly select another bit in the second substring, say the jth bit, such that $1 \leq j < N$. Swapping all the bits to the right of the selected bits in the two mating individuals results in two new individuals:

$S_3 = (B_{11}(1) \dots B_{11}(k)\, B_{21}(k+1) \dots B_{21}(n) \boldsymbol{B_{12}(1) \dots B_{12}(j)} B_{22}(j+1) \dots B_{22}(N))$ and

$S_4 = (B_{21}(1) \dots B_{21}(k)\, B_{11}(k+1) \dots B_{11}(n) B_{22}(1) \dots B_{22}(j) \boldsymbol{B_{12}(j+1) \dots B_{12}(N)})$.

Moreover, crossover is carried out at a chosen probability P_c. If crossover takes place, the two children enter the next generation; otherwise, the two parents themselves enter the new generation. The selected individuals are not removed from the population. Instead, they have a chance to be reselected. This takes place until the new generation is filled.

4.3.5 Mutation

During the selection process, ill-performing individuals are barred from entering the next phase; accompanied with it is the possible loss of biodiversity of the population. Such a loss is undesirable since even though the individual represented by the overall bad performance is of little interest to us, certain characteristics the individual possesses may carry valuable information to our search for the optimal solution. Mutation is therefore introduced to preserve the biodiversity. It is done by occasionally flipping a 0 to a 1 or vice versa. Like crossover, mutation also takes place at a chosen probability. The probability of a given bit being flipped, which is also known as the mutation probability, is P_m.

4.3.6 Elitism

Elitism is adopted in our GA to help improve convergence. It is applied right after the mutation operation. In the first generation, we make a copy of the fittest individual (one that has the highest fitness value), which we call the elite member. In subsequent generations, we compare the elite member with the best individual of the current generation. If the elite member has a

higher fitness value than the best individual of the current generation, we replace a randomly selected individual of the current generation with the elite member and the elite member remains the same. Otherwise, the best individual of the current generation becomes the new elite member.

The flowchart in Figure 4-3 summarizes the main steps involved in a GA.

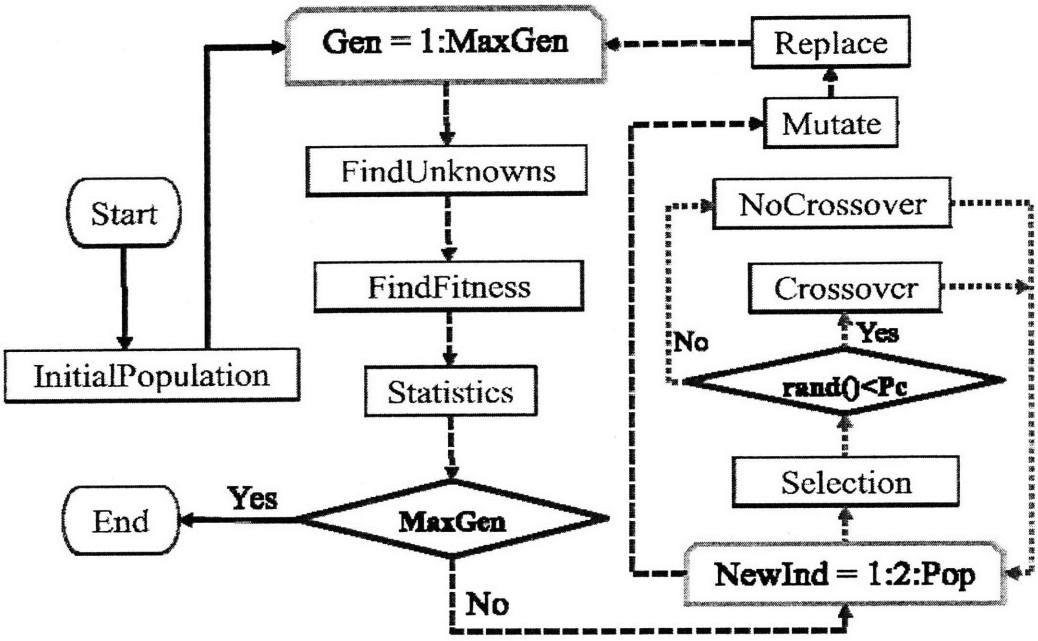


Figure 4-3 Flowchart for main steps of a GA

4.4 Chapter Summary

In this chapter, we review the past works that have been done on RCPSPs. Our own optimization problem is formally formulated to address aspects that are less studied in existing literatures. The optimization method, genetic algorithm, is introduced. Problem specific implementation details are also discussed.

CHAPTER 5
RESULTS AND DISCUSSION

In this chapter, we present the results obtained using our simulation-optimization model in solving several numerical examples. In each example, two sub-problems which only differ by the magnitudes of rework probabilities, are solved individually. Our simulation model as well as our optimization formulation are different from those found in existing literature. The exact optimal solutions to our newly defined problems are unknown. Therefore we assess the quality of our solutions to our problems based on comparisons. For every numerical example, we first compare our solution to the known optimal solution of a special case of our problem. Moreover, we compare our solution with solutions that are obtained using single heuristic rules. From these comparisons we show that the performance of our model is good and our approach is promising in modeling and optimizing real project scheduling problems. We also dedicate one whole section to examine a small example in detail, which sheds light on what constitutes a good solution.

5.1 Choice of Test Problems

There are several sets of test problems that are commonly used by researchers of RCPSPs for the evaluation of solution procedures. Among them, the set of benchmark instances generated by a standard project generator ProGen [15], which was developed by Kolisch and Sprecher, have received much attention. Our test problems originate from the single mode full factorial test set generated by ProGen [16]. Additional modifications to the original problems are made so that they match with our model and problem definitions.

The original test set consists of 480 problems, all of which can be represented by acyclic activity-on-node networks. They are systematically generated by varying three parameters, namely *network complexity* (NC), *resource factor* (RF), and *resource strength* (RS). NC measures the degree of interdependencies among the tasks. It is the average number of non-redundant arcs per node. RF is defined as $RF = \frac{1}{T}\sum_{j=1}^{j=T}\frac{r_j}{m}$, where T is the total number of tasks,

r_j is the number of resource types that task j uses, and m is the total number of resource types available. It is the average number of resource types used by a task expressed as a fraction of the total number of available resource types. RS is a measure of the scarcity/availability of the resources. For a particular type of resource, if we define R_{min} as the minimum resource availability level allowing resource feasibility, R_{max} as the peak per-period usage of that resource in the resource dependent earliest start schedule, and R as the actual availability level, then RS satisfies the following relation $R = R_{min} + round(RS \times (R_{max} - R_{min}))$. Table 5-1 summarizes the levels used for these three parameters. [15]

Parameter	Levels			
NC	1.50	1.80	2.10	
RF	0.25	0.50	0.75	1.00
RS	0.20	0.50	0.70	1.00

Table 5-1 Variable parameter settings for the test problems [15]

The problems have 30 tasks each, 1 to 4 resource types, all renewable. Each task has only one execution mode. The optimal total project durations for these problems, as well as the CPU times needed to solve the problems using a branch-and-bound algorithm proposed by Kolisch and Sprecher, have been documented. Smaller RS values correspond to tighter resource constraints, and hence more difficult problems. We have observed that longer CPU times are needed for highly constrained problems, while less constrained problems are solved much faster.

We sort the problems in ascending orders of the CPU times. Two problems within the 40^{th} percentile are selected. They are relatively easy problems. We also select two more difficult problems, which are in the 90^{th} percentile. Even though only a small fraction of the test problems are selected for our simulations, we can see that they represent very different problems since there are great variations in the parameter settings.

Instance Number	NC	RF	RS	CPU time (sec)
J308_5	1.50	0.50	1.00	0.01
J3038_8	2.10	0.50	0.50	0.04
J3045_9	2.10	1.00	0.20	8.32
J3029_1	1.80	1.00	0.20	209.88

Table 5-2 Parameter settings and CPU times for the chosen test problems

The original problems all assume the general RCPSP optimization formulation mentioned in Section 4.2. Our model is richer, and the problem formulation also differs slightly. Therefore modifications are needed to convert the original test problems to those that satisfy our definitions. These modifications include using stochastic task durations instead of fixed task durations, adding feedback dependency relations and the corresponding rework probabilities and rework impacts, adding learning curve effect data. Since we allow resource variation and the use of two work modes, resource variation bound, rework probability variation bounds and task duration variation bounds also need to be added to the original problems.

As mentioned earlier, we assume triangular distributions for the stochastic task durations. The task durations specified in the original problems are used as the MLVs of the triangular distributions. The BCVs and WCVs are set to be 90% and 120% of the MLVs, respectively. In this way, the distribution is skewed towards the longer durations, which represents the "tendency of work to expand to fill available time—and of human nature to relax when ahead—thus making it less likely that activities will finish early, even if they could" [3].

For each test problem, we randomly generate a certain amount of feedback dependency relations. This results in an average of 0.5 increase on the original NCs. The corresponding rework probabilities and rework impacts are also generated randomly. To investigate whether the rework probabilities affect the optimal solutions, we purposely split each problem into two sub-problems, whereby the only difference between the two is the rework probabilities. Sub-problem 2 has rework probabilities that are 1.5 times of those in sub-problem 1.

For each task, we also randomly generate the learning curve effect values. The resource variation bounds, rework variation bounds, and task duration variation bounds are all set to be (−0.1,+0.1). These are all the same for both sub-problems.

5.2 Simulation Results

The parameter settings used in the optimization process for the four test problems are summarized in Table 5-3. Due to time limitation, we evaluate each individual's fitness value based on the distribution of total project durations that is obtained from 1000 simulations. In other words, for each individual of the GA population, we make a Monte Carlo simulation of 1000 runs.

GA population size	40
Max. no. of generations	50
No. of simulation runs for every individual	1000
No. of project stages	3 (10 tasks each)
Crossover probability	0.6
Permutation probability	0.05

Table 5-3 Simulation parameter settings

Our simulation results as well as detailed discussions are presented in the following sections. We use the original instance numbers to name our test problems. For each problem, the two cases having different rework probabilities are termed sub-problem 1 and sub-problem 2, respectively.

5.2.1 Test Problem J3038_8

Figure 5-1 and Figure 5-2 are the GA convergence histories for the two sub-problems of J3038_8.

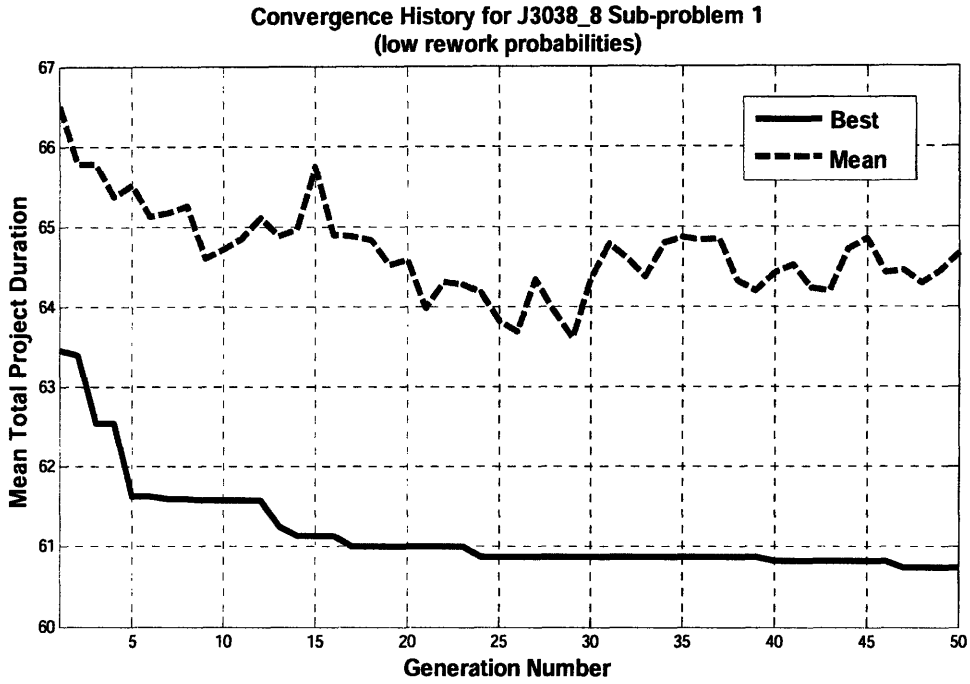


Figure 5-1 Convergence history for J3038_8 sub-problem 1

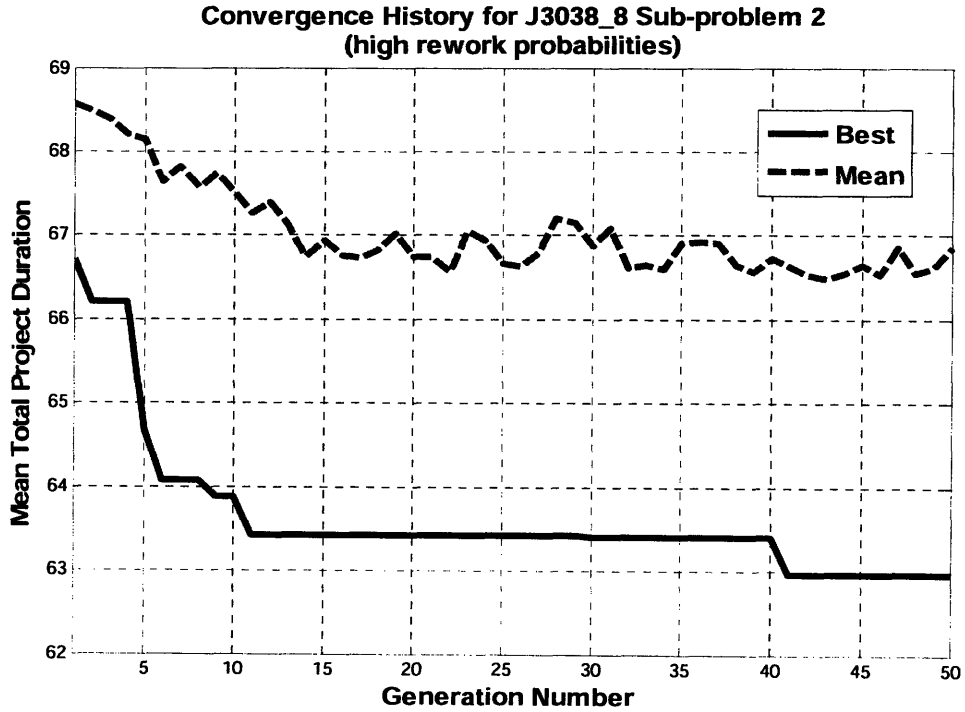


Figure 5-2 Convergence history for J3038_8 sub-problem 2

Extracting the optimal weights and work modes suggested by the GA solutions, we run the Monte Carlo simulations again, doubling the number of runs for each simulation to obtain a truly stable distribution of the total project durations. The distributions for the two sub-problems are shown in Figure 5-3 and Figure 5-4.

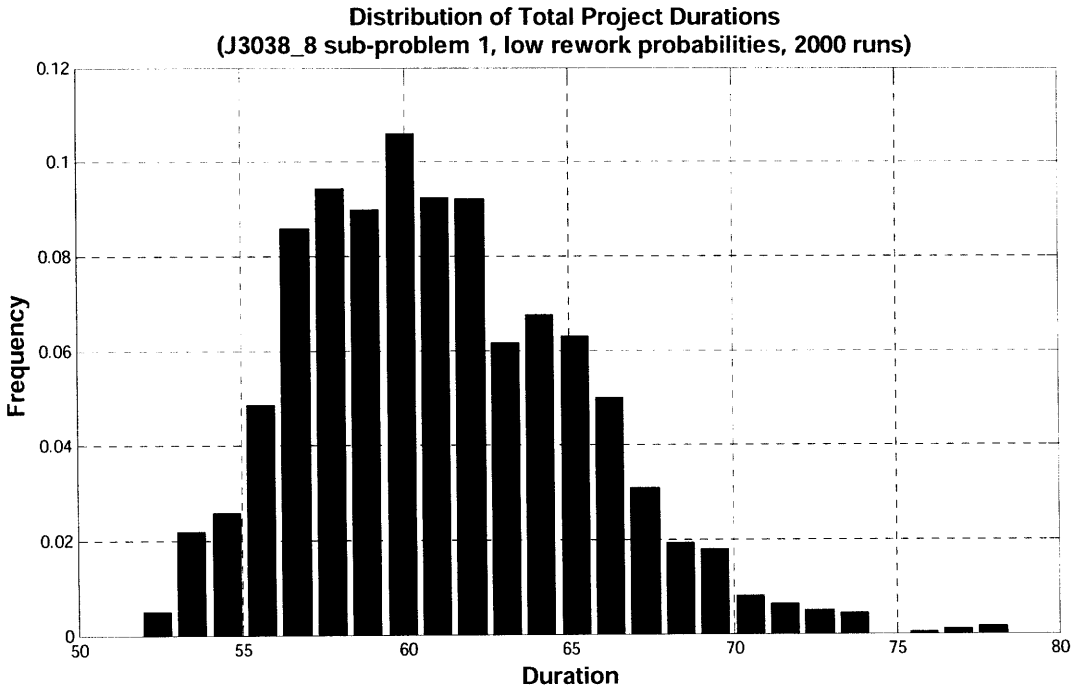


Figure 5-3 Distribution of total project durations for J3038_8 sub-problem 1

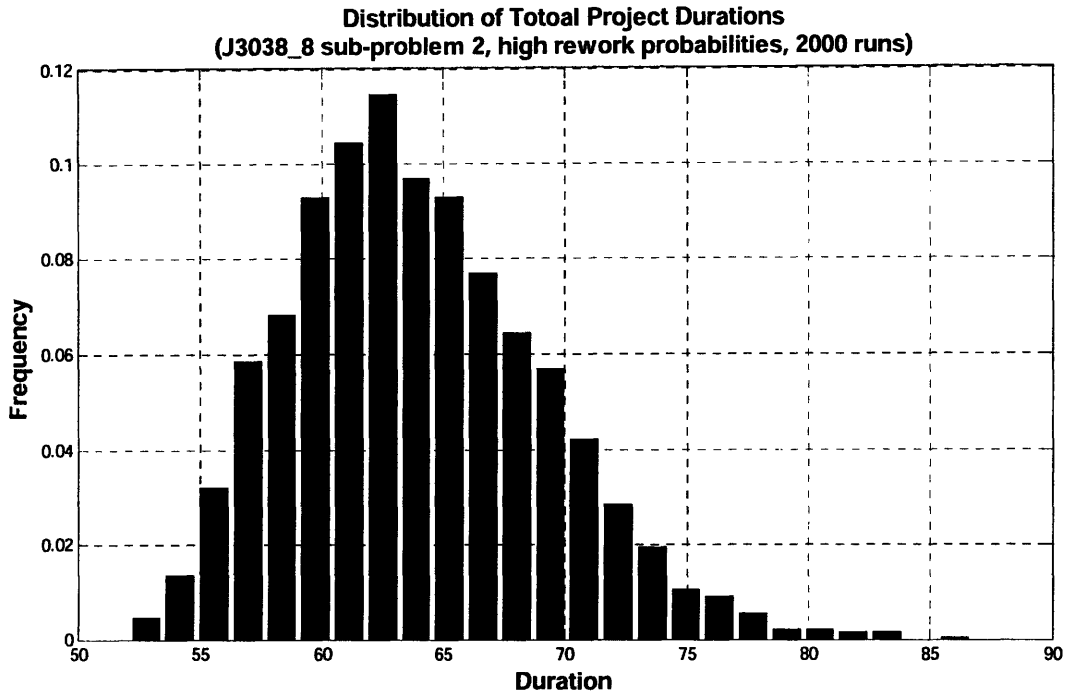


Figure 5-4 Distribution of total project durations for J3038_8 sub-problem 2

We summarize the statistics for the two Monte Carlo simulations in Table 5-4. It is observed that the simulation result of sub-problem 2 has slightly higher mean value and standard deviation than those of sub-problem 1. This is logical since the rework probabilities of sub-problem 2 are higher. Higher rework probabilities means a larger amount of rework is expected. This lengthens the total project durations in general, and results in greater variations from one simulation run to another.

#	min. value	max. value	mean value	standard deviation
1	51.8814	78.4966	61.1171	4.236
2	52.1301	86.738	63.9059	5.19

Table 5-4 Monte Carlo simulation statistics for GA solutions to J3038_8

The exact optimal solutions to the new (sub-) problems are not known to us. Therefore we cannot assess the quality of our GA solutions directly. We assess their qualities through

comparisons instead. Since the original problem is a special case of the new problems (no rework is generated, task durations take the MLVs, nominal amounts of resources are allocated), if the GA solutions perform well for the new problems, they should also perform well on the original problems. Therefore our first comparison is between the optimal solution to the original problem and the GA solutions applied on the original problem.

The 2nd and 3rd columns of Table 5-5 shows such a comparison. We adopt only the task sequences suggested by the GA solutions to solve the original problem and obtain the values in the 3rd column. They are exactly the same as the original optimal solutions.

Solution #	Optimal solution to the original problem	GA solution applied to the original problem	GA solution applied to the original problem (rework added)	GA solution applied to the original problem (two work modes)	GA solution to the sub-problems
1	61	61	62.3912	56.9725	61.1171
2	61	61	64.9941	56.8794	63.9059

Table 5-5 Comparison between GA solutions and original optimal solution for J3038_8

Table 5-5 also shows the transition of the total project durations when the problem evolves from the original problem to our new problems. In the 4th column, rework is added to the original problem, and GA task sequences are used. The total project durations become longer as expected. In the 5th column, no rework is generated, but resource variations and different work modes are allowed. In this case, the time needed is even shorter than the original optimal value. This is because there is a possible reduction in a task's rework probability or duration if more than nominal amounts of resources are allocated to the task. It tells us that if situation permits, resources can and should be traded for time. The last column shows the GA solutions to our exact new problems.

Through these comparisons we can see that the scheduling schemes found by GA are indeed good. Even though we have changed the problem settings from one to another (represented by the various columns of Table 5-5), the GA solutions are robust enough to perform well in all the settings.

In our formulation of the simulation model, we have proposed that our approach of weighted sum of several heuristics is better than using single heuristic rules. To verify our proposition, our second comparison is between our GA solutions with solutions obtained using single heuristic rules.

We examine the performance of four heuristic rules, namely *shortest operation first* (SOF), *maximum critical time first* (MCTF), *longest operation first* (LOF), and *first come first serve* (FCFS). Just as a side note, our FCFS simply means to follow the task sequence as listed in the DSM, instead of anything that is normally used in queuing theory.

Since our model assumes two different work modes when resource levels vary, we would also like to explore whether the work modes found by GA are desirable. Therefore for each heuristic rule we test four cases: using work modes suggested by GA solutions, entirely P modes, entirely T modes, and random modes. The results are summarized in Table 5-6 and Table 5-7.

	SOF				MCTF			
	min	max	mean	sd	min	max	mean	sd
GAmode	52.2086	75.7855	61.7826	4.1226	53.9337	83.7178	63.6374	4.2966
Pmode	55.607	84.5068	64.7093	4.3648	56.5267	86.0247	67.5942	4.8074
Tmode	52.1921	77.6021	61.6796	4.1624	52.7218	76.9956	63.5332	4.1817
randmode	53.3851	81.1542	63.6271	4.4732	53.9255	79.2962	65.8879	4.5462
	LOF				FCFS			
	min	max	mean	sd	min	max	mean	sd
GAmode	56.3876	82.7853	65.851	4.3263	53.7666	79.4464	63.6599	4.1526
Pmode	57.8691	84.5983	68.6228	4.261	56.1837	89.071	65.8968	4.248
Tmode	56.3876	83.5436	65.8624	4.2499	53.8149	77.4003	63.6111	4.1479
randmode	57.0674	82.7699	67.1575	4.3099	54.9775	79.157	64.8398	4.265
GA solution					51.8814	78.4966	61.1171	4.236

Table 5-6 Comparison between GA solution and solutions using single heuristic rules (J3038_8 sub-problem 1)

	SOF				MCTF			
	min	max	mean	sd	min	max	mean	sd
GAmode	52.2241	86.5321	64.6516	5.0068	53.8406	87.6342	66.4608	4.9735
Pmode	56.3021	88.1134	66.7535	4.8784	56.5652	89.8451	69.7611	5.0743
Tmode	52.5539	85.5427	64.668	5.0956	54.4909	90.8322	66.5568	5.078
randmode	54.2417	84.5621	65.8218	4.9285	54.8974	88.7596	68.1007	5.1043
	LOF				MCTF			
	min	max	mean	std	min	max	mean	sd
GAmode	56.9258	89.1966	68.8627	5.1608	54.1221	87.9711	65.7576	4.9826
Pmode	59.0507	92.8053	70.535	4.9278	56.151	84.558	67.2968	4.6336
Tmode	56.2343	86.4135	68.8663	5.1531	54.0385	91.543	65.8004	5.0817
randmode	57.9411	88.2962	69.8243	5.0219	55.2993	85.5009	66.3749	4.742
GA solution					52.1301	86.738	63.9059	5.19

Table 5-7 Comparison between GA solution and solutions using single heuristic rules (J3038_8 sub-problem 2)

From Table 5-6 and Table 5-7 we can see that the GA solutions are better than all the other solutions found using single heuristic rules. The difference is more obvious for sub-problem 2, where the rework probabilities are higher.

The work modes suggested by GA also perform much better than either random modes or entirely P modes. Using entirely T modes outperforms using work modes found by GA occasionally, but the differences are marginal.

We observe in this problem that the tasks are often assigned amounts of resources that are more than their nominal resource requirements, even though parallelization of several tasks cannot be realized due to the limited total amounts of resources available. In this problem, we assume that both the rework probability variation bounds and the task duration variation bounds are $(-0.1,+0.1)$. This probably gives the T mode an advantage over the P mode, since using the T mode results in direct reductions in task durations and hence total project durations, whereas using the P mode only reduces the rework probabilities, which does not reduce the total project

durations as significantly as using the T modes. We will come back to this point again in Section 5.3.

From the two comparisons we have made earlier, we can conclude that the GA solutions to this test problem are optimal and robust.

We also examine the relation between rework probabilities and the GA solutions. Even though the weights obtained in the GA solutions to the two sub-problems differ significantly, they result in very similar task sequences, and very close mean total project durations. The difference in mean total project durations should be due to the differences in rework probabilities. Moreover, one of the ranking factors, the critical time of a task, always receives a significant weight, whether the rework probabilities are low or high.

5.2.2 Test Problem J308_5

Problem J308_5 is also an easy problem. Observations about its GA solutions are very similar to those discussed in Section 5.2.1. We shall present the relevant graphs and data here, but omit the step-by-step analysis.

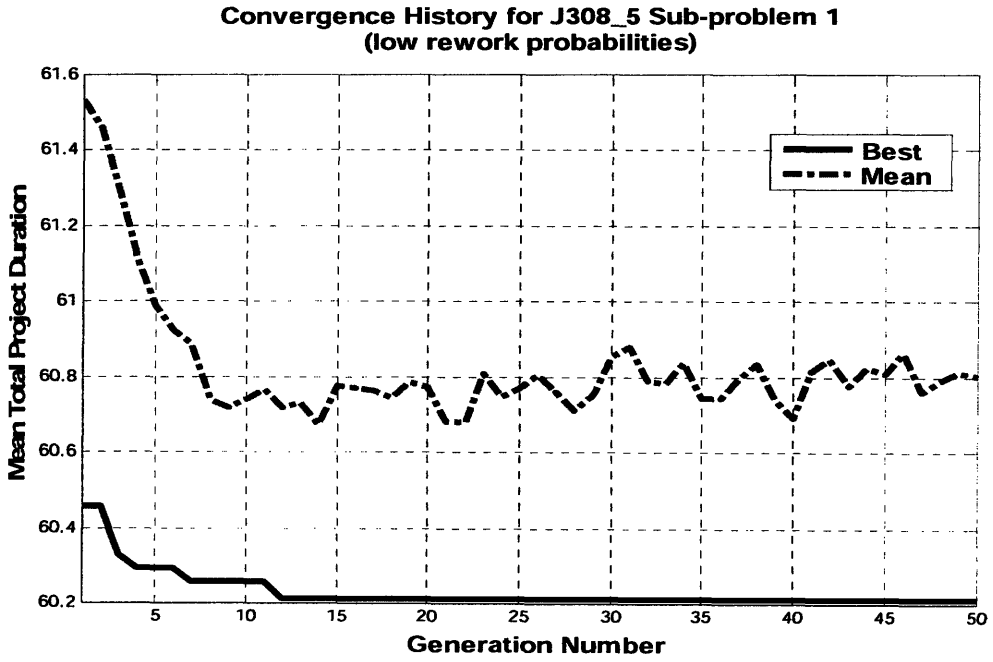


Figure 5-5 Convergence history for J308_5 sub-problem 1

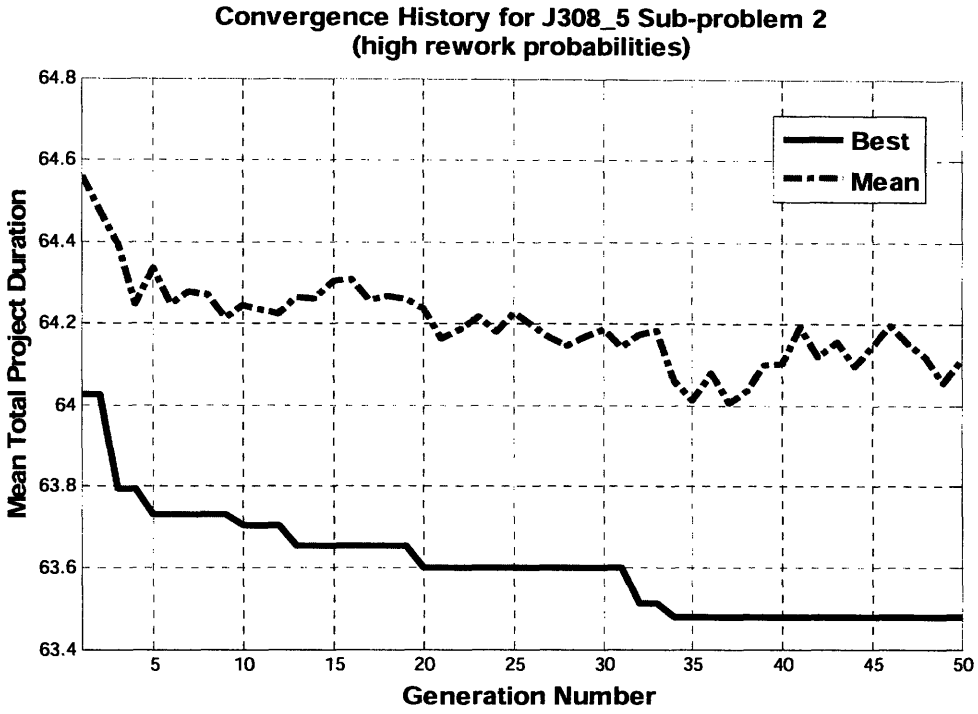


Figure 5-6 Convergence history for J308_5 sub-problem 2

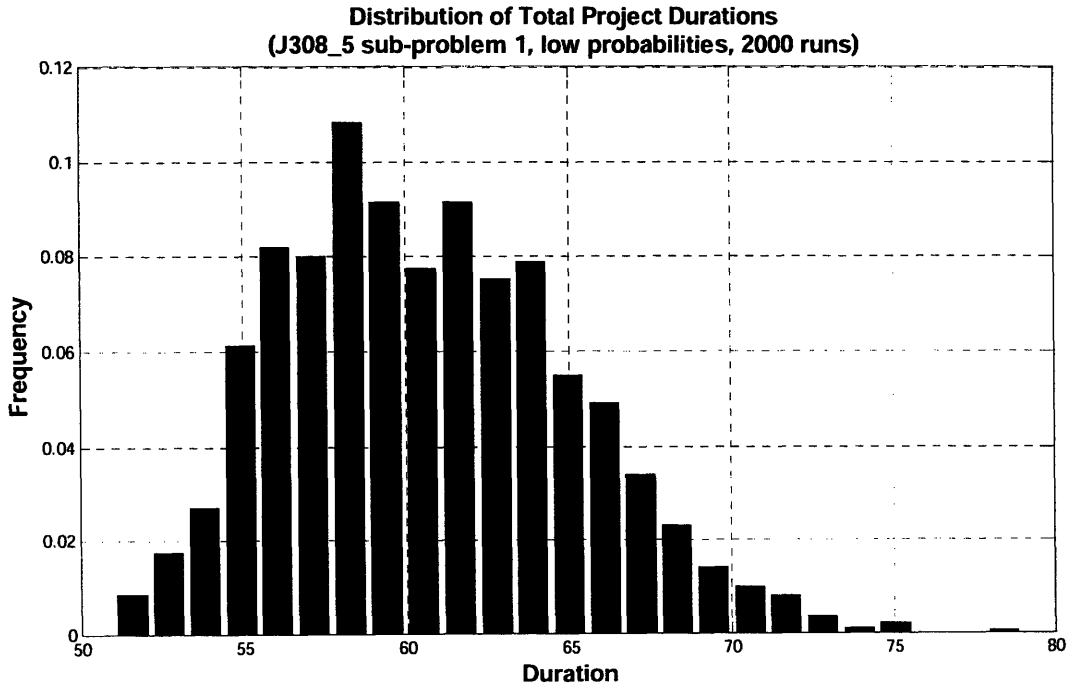


Figure 5-7 Distribution of total project durations for J308_5 sub-problem 1

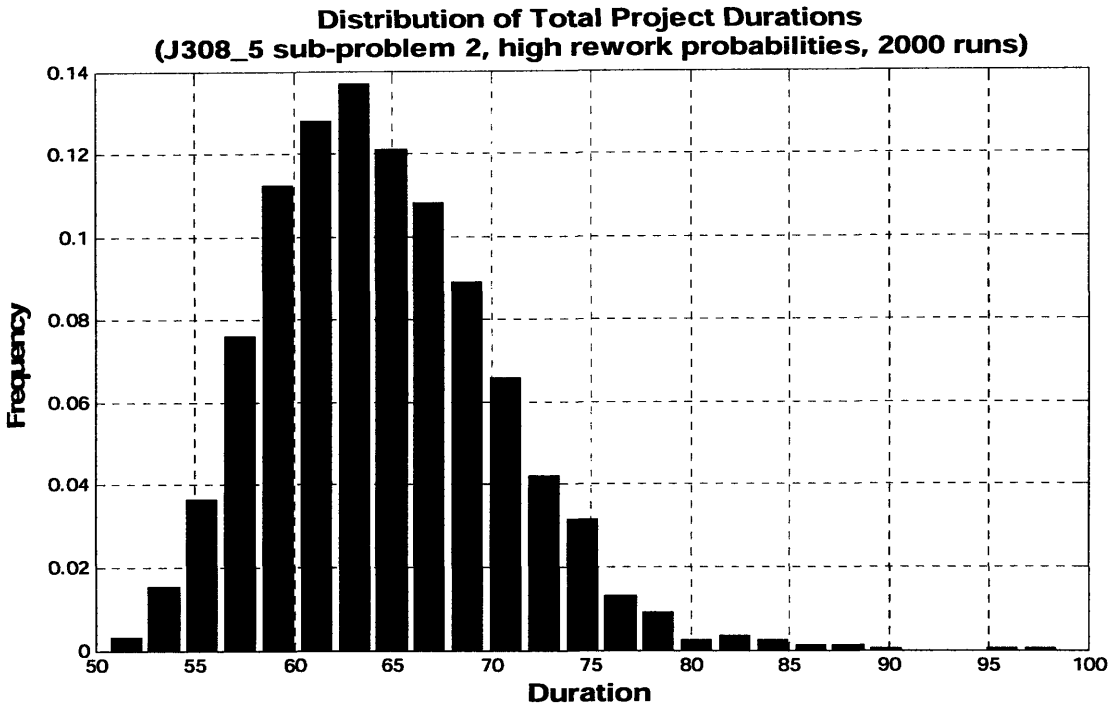


Figure 5-8 Distribution of total project durations for J308_5 sub-problem 2

#	min. value	max. value	mean value	standard deviation
1	51.016	78.9529	60.6717	4.3911
2	50.5673	98.5385	64.4611	5.7992

Table 5-8 Monte Carlo simulation statistics for GA solutions to J308_5

Solution #	Optimal solution to the original problem	GA solution applied to the original problem	GA solution applied to the original problem (rework added)	GA solution applied to the original problem (two work modes)	GA solution to the sub-problems
1	58	58	60.8876	55.0596	51.016
2	58	58	64.4536	55.0596	50.5673

Table 5-9 Comparison between GA solutions and original optimal solution for J308_5

	SOF				MCTF			
	min	max	mean	sd	min	max	mean	sd
GAmode	50.3336	75.2887	60.5761	4.2224	50.5163	79.2181	60.643	4.4224
Pmode	52.2805	78.2501	62.5795	4.4185	52.6016	79.5903	62.3679	4.4931
Tmode	50.3336	79.7511	60.7065	4.4524	50.7448	80.5	60.5997	4.3517
randmode	51.1114	79.6227	61.5876	4.4045	51.5565	77.3947	61.5449	4.4244
	LOF				FCFS			
	min	max	mean	sd	min	max	mean	sd
GAmode	50.3132	85.5507	60.7831	4.5317	50.571	80.5	60.5627	4.3309
Pmode	52.5645	80.0917	62.7315	4.58	52.5095	76.3119	62.332	4.4001
Tmode	49.8997	80.5	60.713	4.4767	50.4143	76.265	60.44	4.4689
randmode	51.1114	78.8492	61.7229	4.5717	51.4482	77.9845	61.2754	4.3577
GA solution					51.016	78.9529	60.6717	4.3911

Table 5-10 Comparison between GA solution and solutions using single heuristic rules (J308_5 sub-problem 1)

	SOF				MCTF			
	min	max	mean	sd	min	max	mean	sd
GAmode	51.2384	86.6918	64.4469	5.3078	51.5741	87.9758	64.5246	5.4216
Pmode	52.412	84.7079	64.7348	4.9104	53.5087	89.5643	64.8743	5.272
Tmode	51.4607	89.115	64.6008	5.6742	51.0248	87.3295	64.0902	5.4305
randmode	52.2018	81.4554	64.4437	5.0838	51.7654	81.5341	64.2313	5.046
	LOF				FCFS			
	min	max	mean	sd	min	max	mean	sd
GAmode	52.6333	88.1188	64.7049	5.6877	50.5161	87.9758	64.6157	5.5504
Pmode	52.6491	86.8472	65.7741	5.2321	53.134	84.4232	64.8436	5.0232
Tmode	51.2853	93.2357	64.9848	5.7875	51.0588	92.3698	64.3625	5.5767
randmode	52.0125	85.4098	65.1446	5.2821	52.9657	84.08	64.438	5.1188
GA solution					50.5673	98.5385	64.4611	5.7992

Table 5-11 Comparison between GA solution and solutions using single heuristic rules (J308_5 sub-problem 2)

5.2.3 Test Problems J3045_9 and J3029_1

J3045_9 and J3029_1 are more difficult problems. Their results have many similarities. We present their respective graphs and data in this section and leave the discussions about the results to the following section.

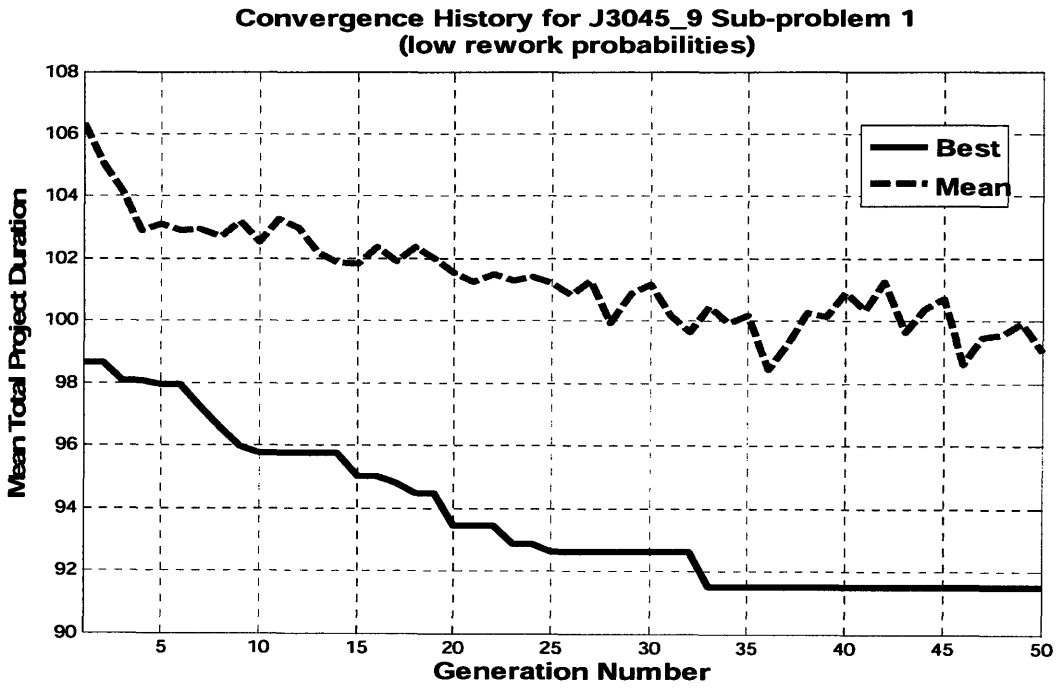


Figure 5-9 Convergence history for J3045_9 sub-problem 1

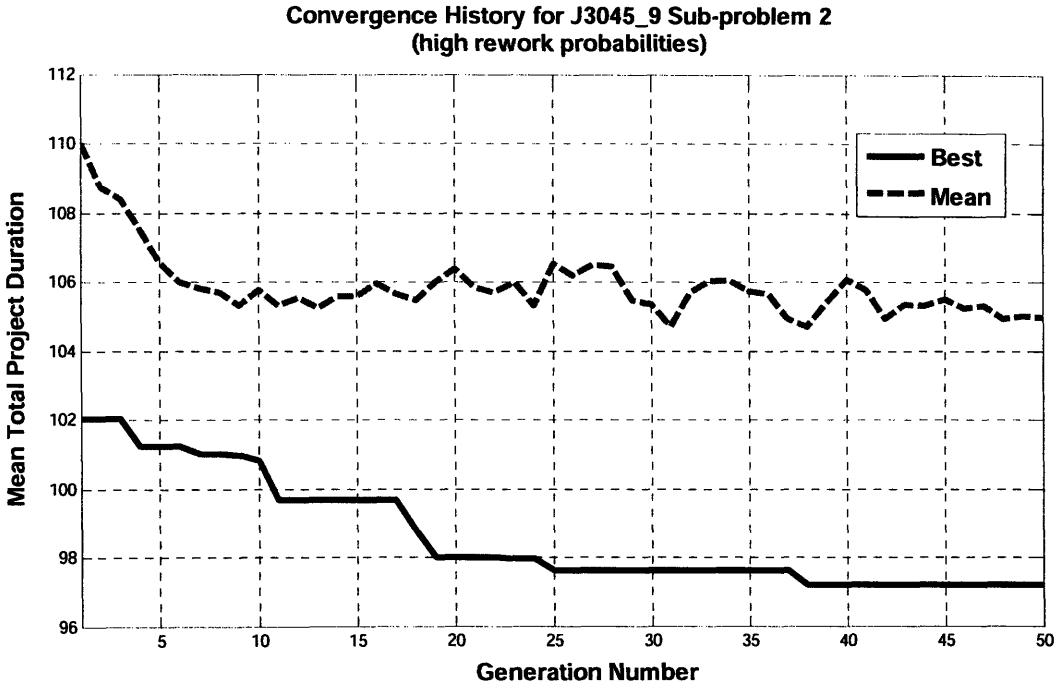


Figure 5-10 Convergence history for J3045_9 sub-problem 2

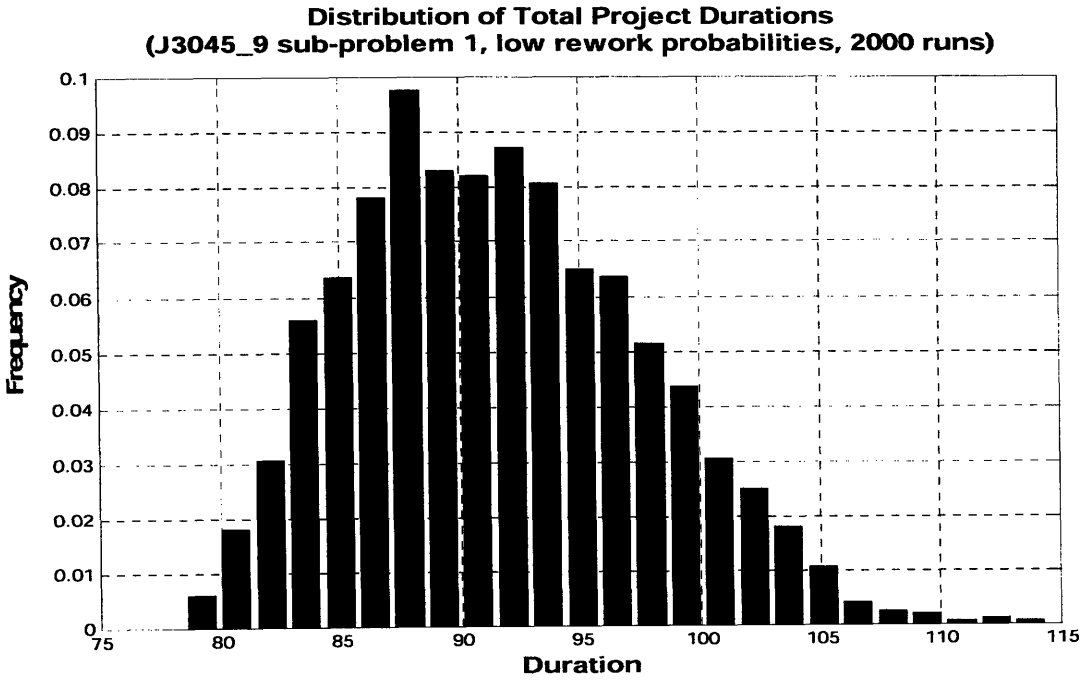


Figure 5-11 Distribution of total project durations for J3045_9 sub-problem 1

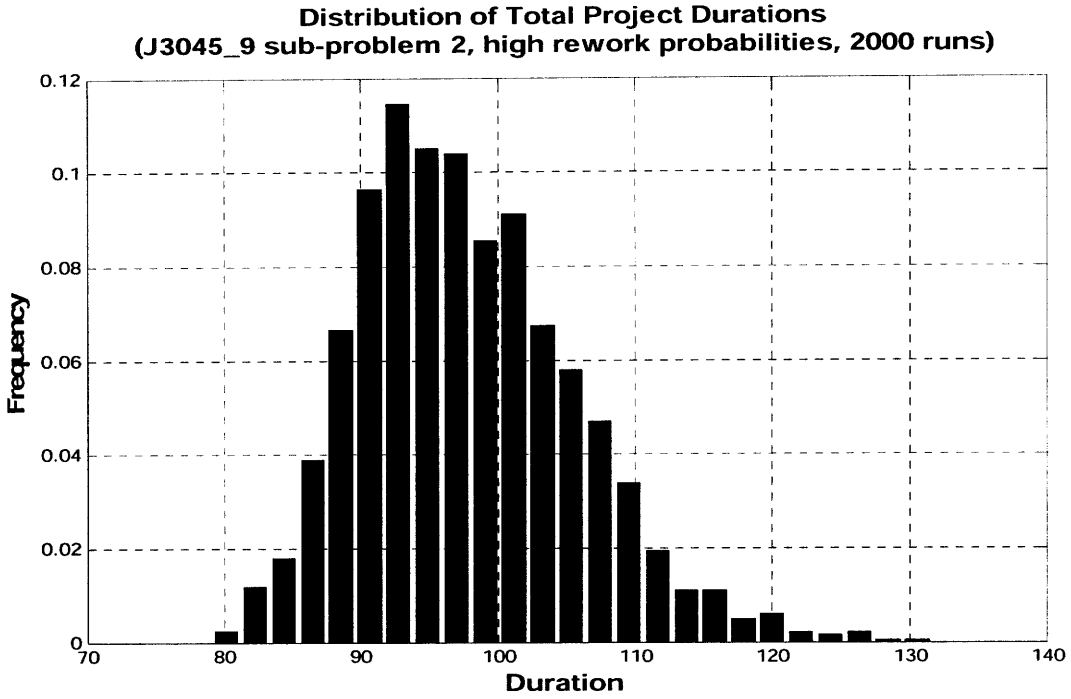


Figure 5-12 Distribution of total project durations for J3045_9 sub-problem 2

#	min. value	max. value	mean value	standard deviation
1	78.4431	114.4648	91.6009	6.035
2	79.1435	131.6313	97.7841	7.845

Table 5-12 Monte Carlo simulation statistics for GA Solutions to J3045_9

Solution #	Optimal solution to the original problem	GA solution applied to the original problem	GA solution applied to the original problem (rework added)	GA solution applied to the original problem (two work modes)	GA solution to the sub-problems
1	82	94	96.6621	84.7667	91.6009
2	82	95	103.0415	85.6644	97.7841

Table 5-13 Comparison between GA solutions and original optimal solution for J3045_9

	SOF				MCTF			
	min	max	mean	sd	min	max	mean	sd
GAmode	89.5777	127.9409	105.5853	6.8276	77.7229	111.1016	92.2364	6.012
Pmode	93.4147	135.6707	112.2246	7.3013	82.486	119.7409	96.2005	6.3827
Tmode	89.1413	130.2379	105.4481	6.8109	77.9797	113.116	92.3391	6.1799
randmode	90.526	131.9924	108.9288	7.114	79.8923	113.6441	94.555	6.136
	LOF				FCFS			
	min	max	mean	sd	min	max	mean	sd
GAmode	89.8677	126.0546	105.2644	6.7765	80.8937	114.6601	94.4471	5.9587
Pmode	93.0764	132.7752	109.8313	7.0776	84.2295	121.0262	98.8317	6.2132
Tmode	89.1959	127.3957	105.1533	6.8085	81.3006	117.2937	94.3067	5.9797
randmode	90.6079	134.1722	107.5195	6.9557	81.9027	116.1952	96.6235	6.2108
GA solution					78.4431	114.4648	91.6009	6.035

Table 5-14 Comparison between GA solution and solutions using single heuristic rules (J3045_9 sub-problem 1)

	SOF				MCTF			
	min	max	mean	sd	min	max	mean	sd
GAmode	91.0028	142.6409	110.2376	8.2927	80.2027	137.2831	97.1293	7.4418
Pmode	97.4149	147.8004	115.602	8.0331	82.4957	127.0881	99.8453	6.9934
Tmode	90.8388	153.6126	109.8512	7.8105	78.808	127.4814	97.1139	7.3561
randmode	93.9196	145.1494	112.7783	8.1293	79.7105	127.7024	98.2691	7.2183
	LOF				FCFS			
	min	max	mean	sd	min	max	mean	sd
GAmode	90.5892	138.2925	110.4417	8.1442	82.4436	130.9482	98.9154	7.6141
Pmode	94.1031	146.4337	113.7621	7.9336	86.0264	129.412	102.0621	7.3579
Tmode	92.2965	151.153	110.1276	8.3061	81.3006	135.3962	98.8087	7.4705
randmode	91.4323	145.8943	111.8695	8.1895	81.967	131.2319	100.4164	7.4601
GA solution					79.1435	131.6313	97.7841	7.845

Table 5-15 Comparison between GA solution and solutions using single heuristic rules (J3045_9 sub-problem 2)

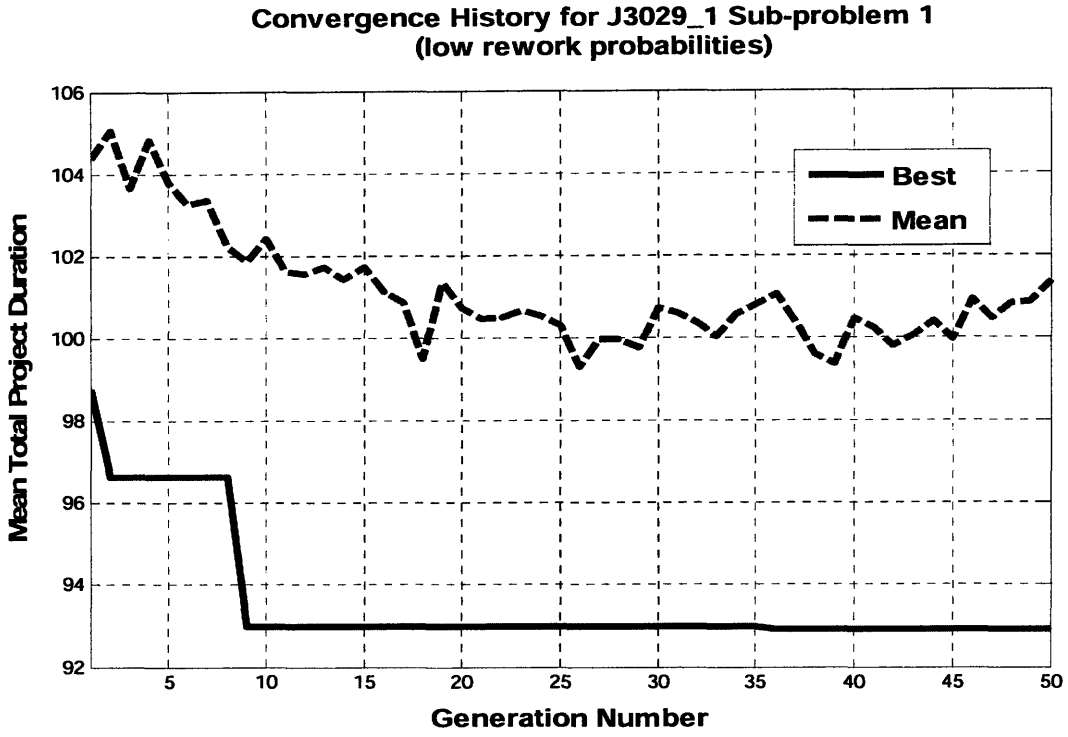


Figure 5-13 Convergence history for J3029_1 sub-problem 1

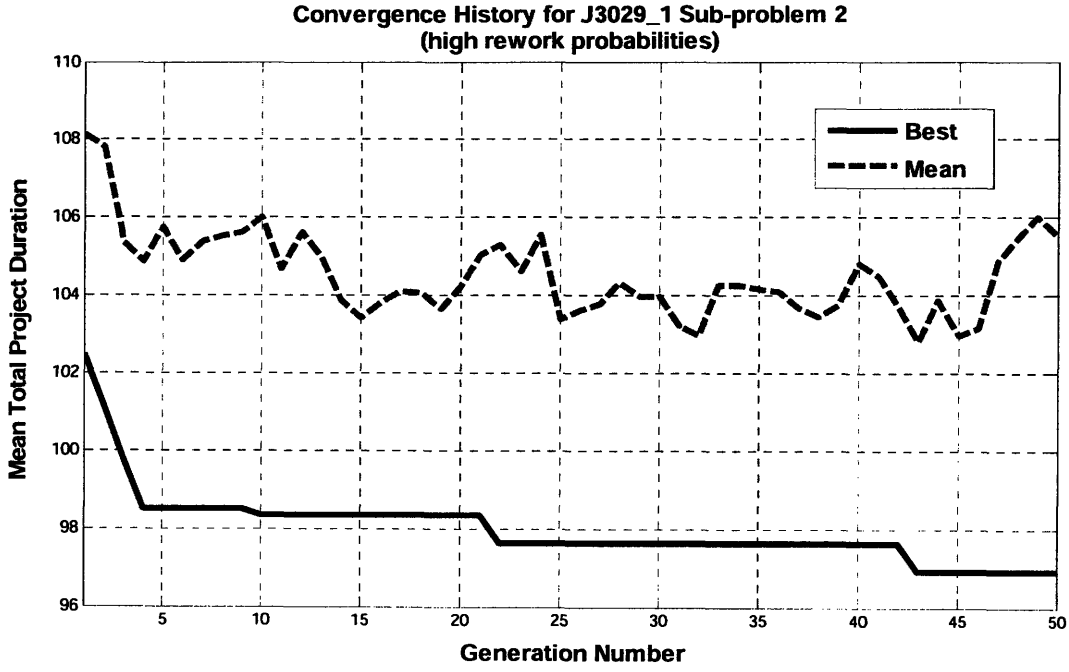


Figure 5-14 Convergence history for J3029_1 sub-problem 2

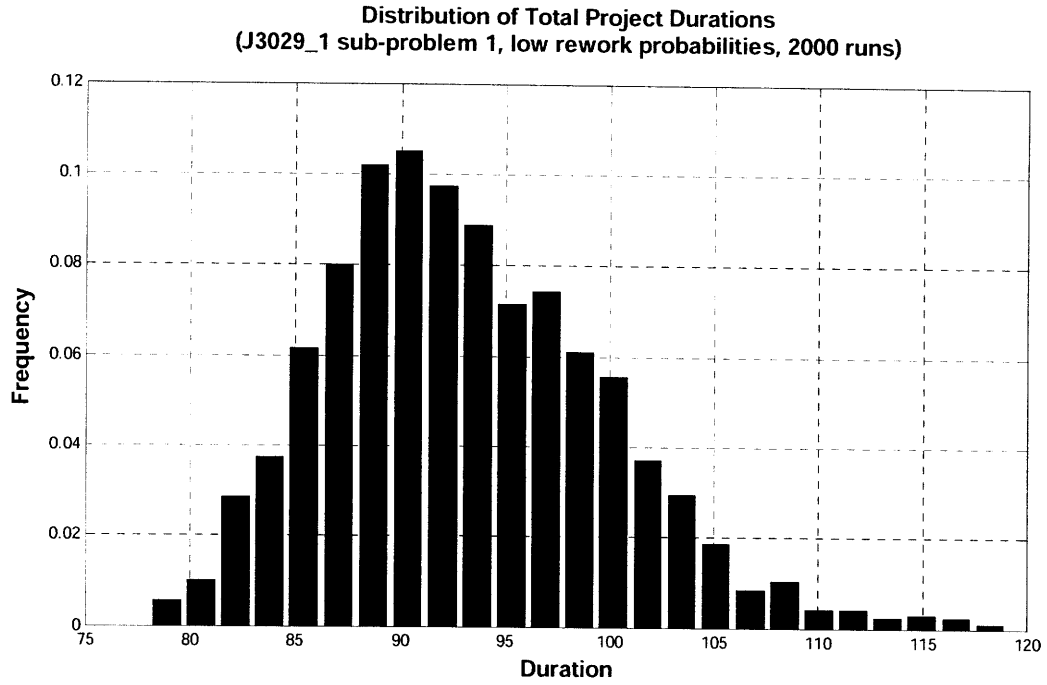


Figure 5-15 Distribution of total project durations for J3029_1 sub-problem 1

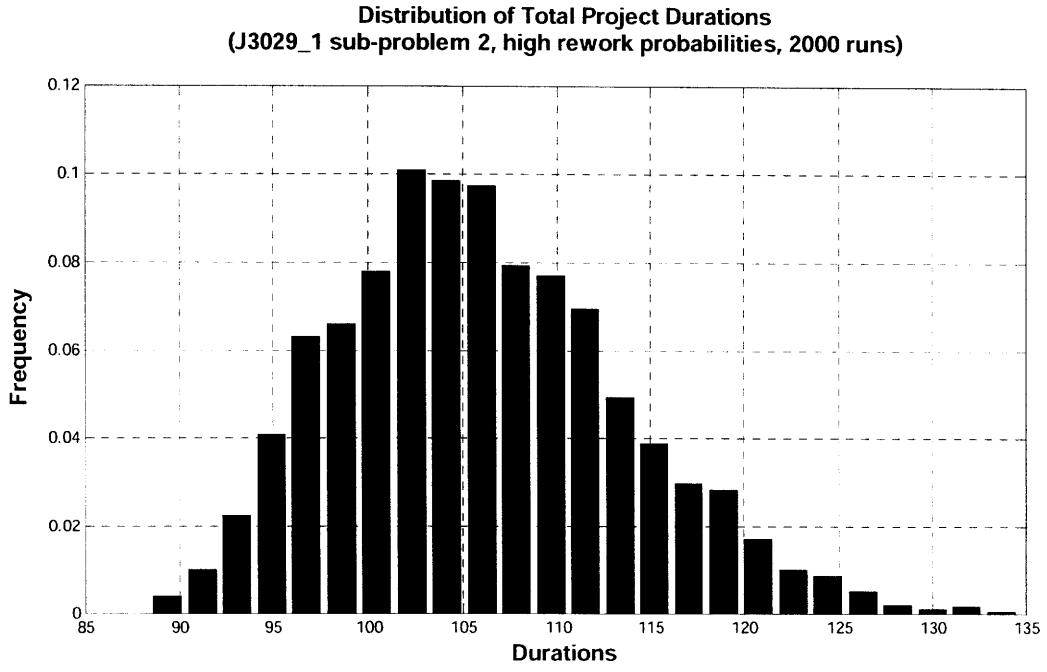


Figure 5-16 Distribution of total project durations for J3029_1 sub-problem 2

#	min. value	max. value	mean value	standard deviation
1	78.071	119.06	92.925	6.6728
2	88.433	134.6	106	7.556

Table 5-16 Monte Carlo simulation statistics for GA Solutions to J3029_1

Solution #	Optimal solution to the original problem	GA solution applied to the original problem	GA solution applied to the original problem (rework added)	GA solution applied to the original problem (two work modes)	GA solution to the sub-problems
1	85	96	98.9926	85.6246	92.925
2	85	104	109.8641	96.8048	106

Table 5-17 Comparison between GA solutions and original optimal solution for J3029_1

	SOF				MCTF			
	min	max	mean	sd	min	max	mean	sd
GAmode	82.6882	116.9772	97.8652	6.3283	78.9263	120.763	93.8584	6.5786
Pmode	88.0841	121.8816	103.1973	6.4669	82.0115	124.422	97.669	6.5236
Tmode	83.3242	118.6115	97.7567	6.1498	78.3546	120.763	94.1465	6.6369
randmode	84.9166	122.2216	100.354	6.4385	81.315	121.1949	95.9317	6.6507
	LOF				FCFS			
	min	max	mean	sd	min	max	mean	sd
GAmode	91.23	129.2906	107.0132	6.8682	80.53	118.0622	94.7392	6.3756
Pmode	96.4485	136.3916	113.3714	7.0944	83.9337	123.7438	98.3438	6.4113
Tmode	90.8144	129.1718	106.8917	6.8579	80.3184	118.0858	94.9498	6.1493
randmode	92.5475	131.6214	110.1581	7.1779	80.2189	117.5645	94.5564	6.1901
GA solution					78.071	119.06	92.925	6.6728

Table 5-18 Comparison between GA solution and solutions using single heuristic rules (J3029_1 sub-problem 1)

	SOF				MCTF			
	min	max	mean	sd	min	max	mean	sd
GAmode	91.138	116.7698	96.9469	3.1551	85.9809	115.5724	94.8824	4.5493
Pmode	88.2631	129.4765	105.5261	7.0763	82.2522	127.1096	100.872	7.3574
Tmode	82.7802	125.3806	100.4768	6.944	80.9523	129.3796	98.0127	7.4661
randmode	85.0754	123.8143	102.9134	7.0829	81.5655	126.43	99.0955	7.3699
	LOF				FCFS			
	min	max	mean	sd	min	max	mean	sd
GAmode	100.4165	124.5411	107.8204	4.0174	88.2648	115.6043	95.1325	3.7611
Pmode	98.4923	142.379	116.4466	7.8912	84.8146	126.2024	101.387	7.1845
Tmode	91.3817	137.7195	110.9153	7.8846	80.9918	123.5948	98.348	6.8783
randmode	95.1519	138.2563	113.4577	7.5257	83.457	126.1373	99.7873	6.9607
GA solution					88.433	134.6	106	7.556

Table 5-19 Comparison between GA solution and solutions using single heuristic rules (J3029_1 sub-problem 2)

5.2.4 Discussion of Results for J3045_9 and J3029_1

For these two problems, we make similar comparisons as we do in Section 5.2.1. In the first comparison with the optimal solutions to the original problems, we observe greater discrepancies between the GA solutions and the original optimal solutions. The average deviation is about 16.5% of the original optimal value, as shown in Table 5-20.

	GA solution	Optimal value	Deviation
J3045_9 sub-problem 1	94	82	14.63%
J3045_9 sub-problem 2	95	82	15.85%
J3029_1 sub-problem 1	96	85	12.94%
J3029_1 sub-problem 2	104	85	22.35%
		Average Deviation	16.44%

Table 5-20 Deviation of GA solutions from optimal solutions to original problems

The worst case is the solution to J3029_1 sub-problem 2. It probably represents a prematurely converged solution, since even the solution to J3029_1 sub-problem 1 performs better on sub-problem 2 than its own solution. GA uses 1000 runs for the Monte Carlo simulation of each individual. The GA convergence history for J3029_1 sub-problem 2 shows that the optimal mean total project duration is about 97. However, the mean total project duration obtained from the simulation with 2000 runs is 9 units longer. This discrepancy suggests that 1000 runs are probably not enough to generate a stable distribution of the mean total project durations. As a result, the fitness values calculated by the GA are not accurate, and the GA terminates prematurely.

M.Wall proposed a different genetic algorithm approach in [24] and tested them on the original set of Kolisch and Sprecher's problems. The performance of his GA is shown in Figure 5-13.

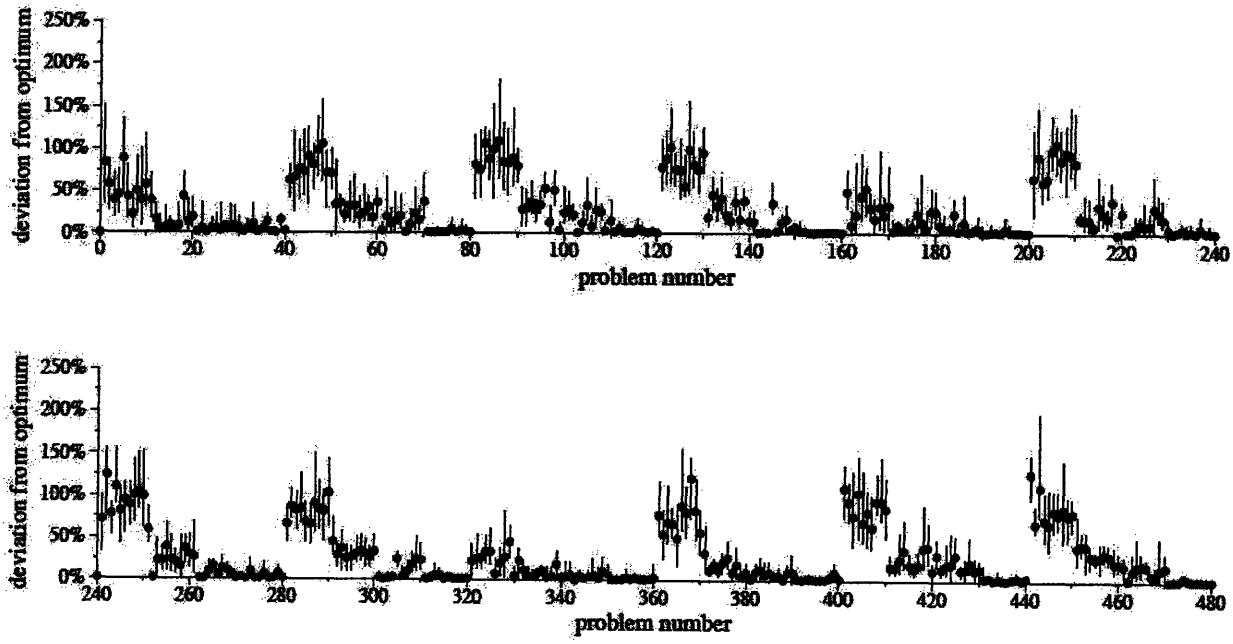
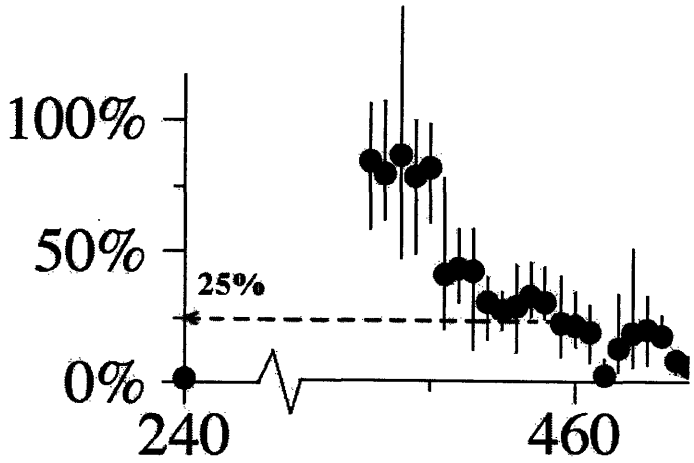


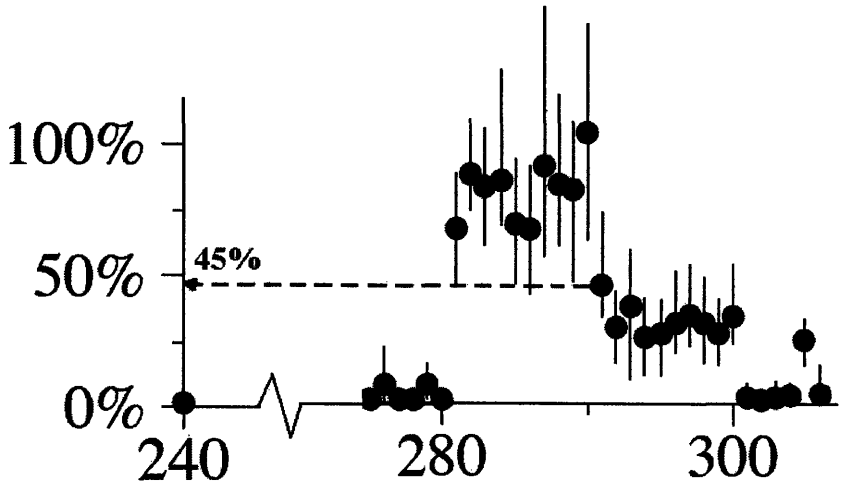
Figure 5-17 Performance of M.Wall's GA on the test problems

Zoom in to problems number 459 (J3045_9) and 291(J3029_1), as shown in Figure 5-18, we can see that the average deviations from the optima are about 25% and 45%, respectively, while large percentages of the problems can be solved to optimality. Our GA solutions, when applied

to the original problems, only deviate from the optimal values by approximately 15%. There is a 10% reduction in the total project durations for these problems.



(a) J3045_9



(b) J3029_1

Figure 5-18 Performance of M.Wall's GA on J3045_9 and J3029_1

The second comparison is between GA solutions to our new problems and solutions obtained using single heuristic rules. GA outperform most of the single heuristic rule methods on the two problems. Only MCTF method obtains solutions that are marginally better occasionally. This is

not surprising since the critical values of the tasks is one ranking factor in our weighted sum approach, and it always receives a heavy weight.

Since our GA is designed to solve the new sub-problems instead of the original problems directly, it is possible that these solutions are just not robust enough to cater to other similar problems. Nevertheless, the sub-optimal performances on the original J3045_9 and J3029_1 show the limitation of our optimization algorithm. One possible reason for GA solution stuck at local minimum points is that the resolution of the weights is not fine enough. Greater resolution may be needed to distinguish the small differences among the ranking factors, hence to rank the tasks correctly. Moreover, even if the resolutions of the weights are high enough, due to the coarse movements brought about by the crossover and mutation operations, it is difficult for the algorithm to locate the exact optimal point in the search space. In other words, the current GA does not perform well if the optimal point lies at the bottom of a deep narrow valley. Another possible reason for premature convergence is that the problem has many local minima. GA simply settles down at one of them.

There are several possible improvements to the current search algorithm. Firstly, we can strive to get better initial population that are located uniformly in the search space. A better coverage of the search space in the initial generation allows more thorough exploration. Rather than generating the initial population using a random 0-1 generator, we may use the Halton sequence as our initial population, then code the population into binary strings. To prevent premature convergence to a local minimum point, instead of one, we can keep a number of elite members in each generation. Not only that the elite members have high fitness values, but also they represent very different solutions. When the GA solution is about to converge, we can do a local search for all the elite members to see whether any refinement of the weights will result in better solutions. In this way, we have a higher chance of finding optimal solutions.

5.3 Interpretation of GA Solutions

In this section we use a smaller-sized example to illustrate how the GA solutions can be interpreted. The purpose of this section is to see what insights we can gain from the GA solutions apart from the pure numbers that we have displayed in earlier sections.

The example we use in this section consists of 14 tasks. The reason for using a smaller sized project as our example in this section is that the details pertaining to scheduling and resource allocation become more tractable.

The Gantt chart in Figure 5-19 shows the task sequence for the project assuming infinite amount of resources. Rework is randomly generated and shown in the chart as well.

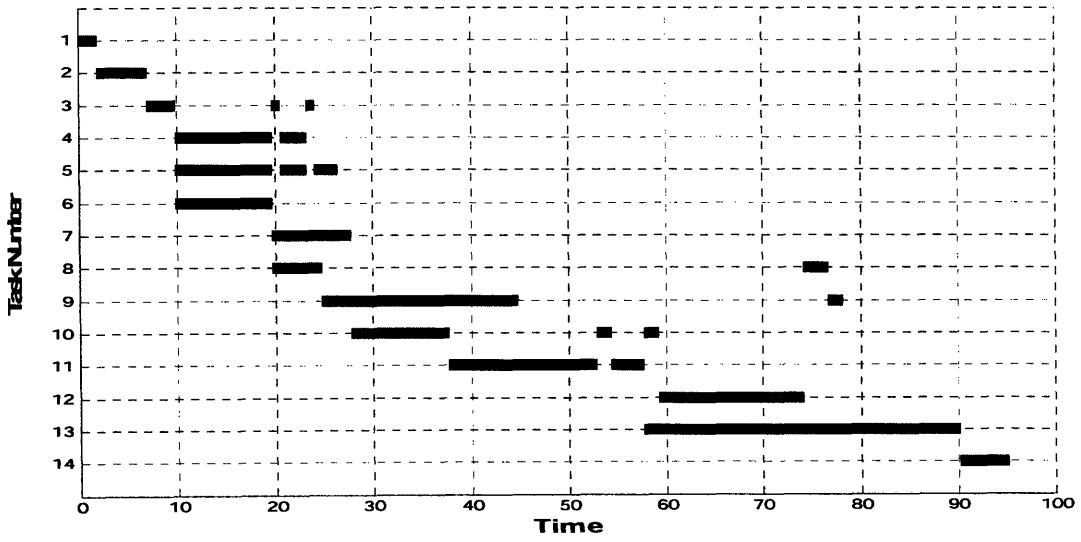


Figure 5-19 Gantt chart of a sample problem (assume infinite resources)

When resource constraints are not imposed, many tasks can be executed in parallel. For example, when the first three tasks are finished, tasks 4, 5, and 6 can start simultaneously. Tasks 9 and 10 can also be executed concurrently after their respective predecessors are finished, since they do not depend on each other. However, after adding the resource constraints, part of the parallelization becomes infeasible since there are not enough resources to start all the tasks that are originally parallel. When there is a resource conflict, we have to make a decision on which tasks are to be delayed. Table 5-21 shows resource constraints for this example. There are four

types of renewable resources, R1, R2, R3, and R4. RA are the resources available. T1 through T14 are the resource requirements for the tasks.

	R1	R2	R3	R4		R1	R2	R3	R4		R1	R2	R3	R3
RA	18	12	14	9	T5	6	7	7	1	T10	1	9	9	8
T1	2	1	7	0	T6	3	5	6	0	T11	0	5	9	3
T2	6	6	1	5	T7	3	9	1	0	T12	1	8	2	7
T3	0	2	9	3	T8	7	3	0	7	T13	1	7	9	9
T4	2	2	1	2	T9	9	7	2	0	T14	9	3	6	1

Table 5-21 Resource constraints for the sample problem

Due to the resource constraints, tasks 4, 5, and 6 can no longer be executed together. The same thing applies to the group consisting of tasks 9 and 10 as well as the group consisting of tasks 12 and 13. The GA solution shows that the optimal sequence is to start tasks 5 and 6 first while delaying task 4. Moreover, task 10 is executed before task 9, so that task 10's immediate successor task 11 can be executed in parallel with task 9. The sequence of tasks 12 and 13 does not really affect the overall time, so either one can be executed first. The Gantt chart for the optimal GA solution is shown in Figure 5-20. This solution is obtained by dividing the tasks into 3 stages of 6 tasks, 6 tasks, and 2 tasks respectively, and hence three sets of weights are used.

Figure 5-21 shows a sub-optimal GA solution obtained using 2 stages of 10 tasks and 4 tasks, respectively. The main difference is that task 9 is chosen to be executed earlier than task 10. Since Task 11 can only start after task 10 is finished, the critical path length of these three tasks is the sum of all the three task durations. However, in the optimal solution, the critical path length is only the sum of two task durations. Compare Figure 5-20 and Figure 5-21, we can see that even though the total amounts of rework generated for both cases are approximately equal, the sub-optimal solution requires 15 units of time more than the optimal solution.

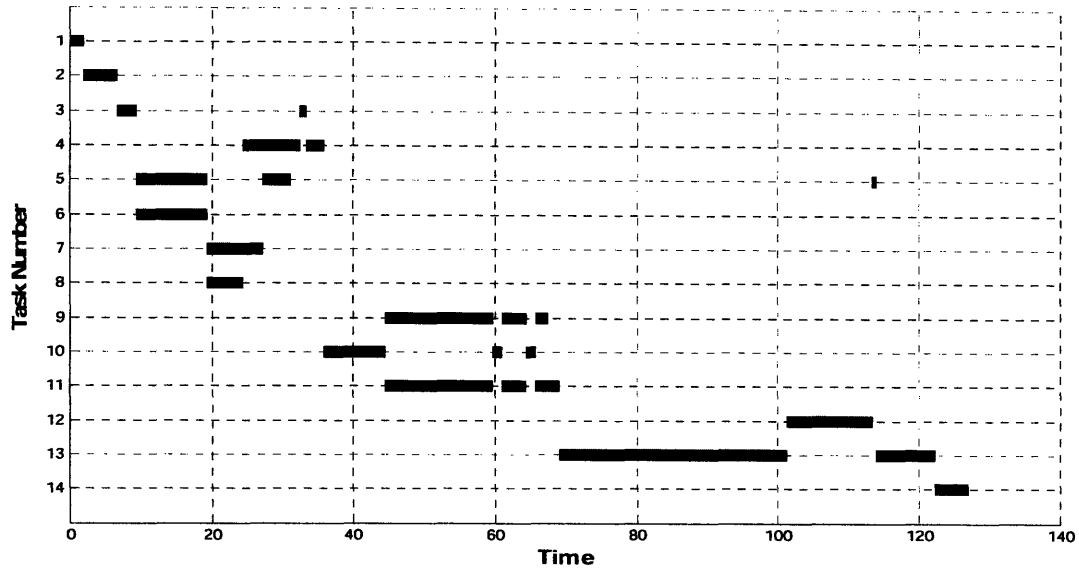


Figure 5-20 Optimal GA solution

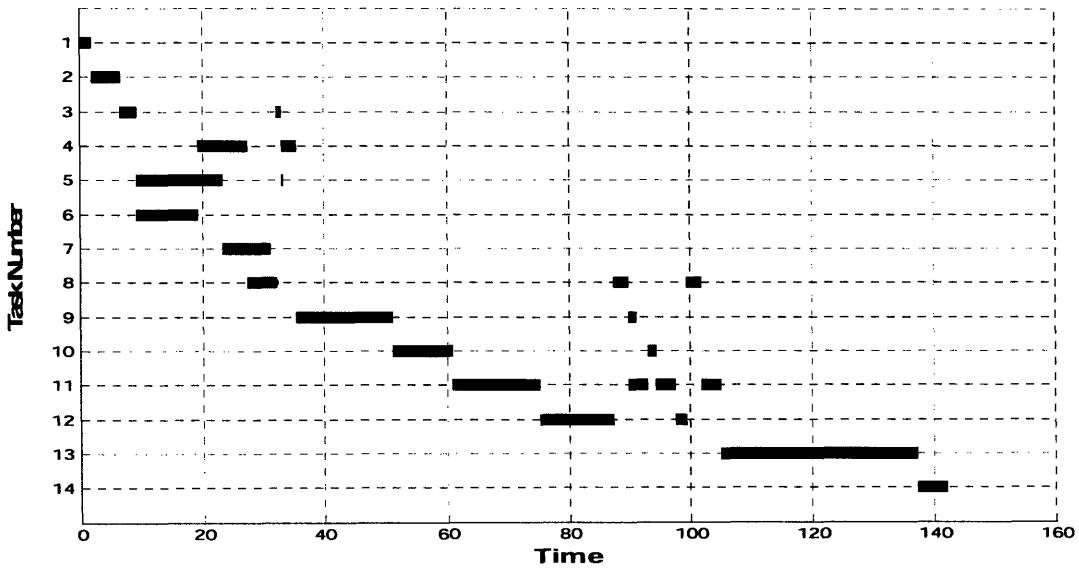


Figure 5-21 Sub-optimal GA solution

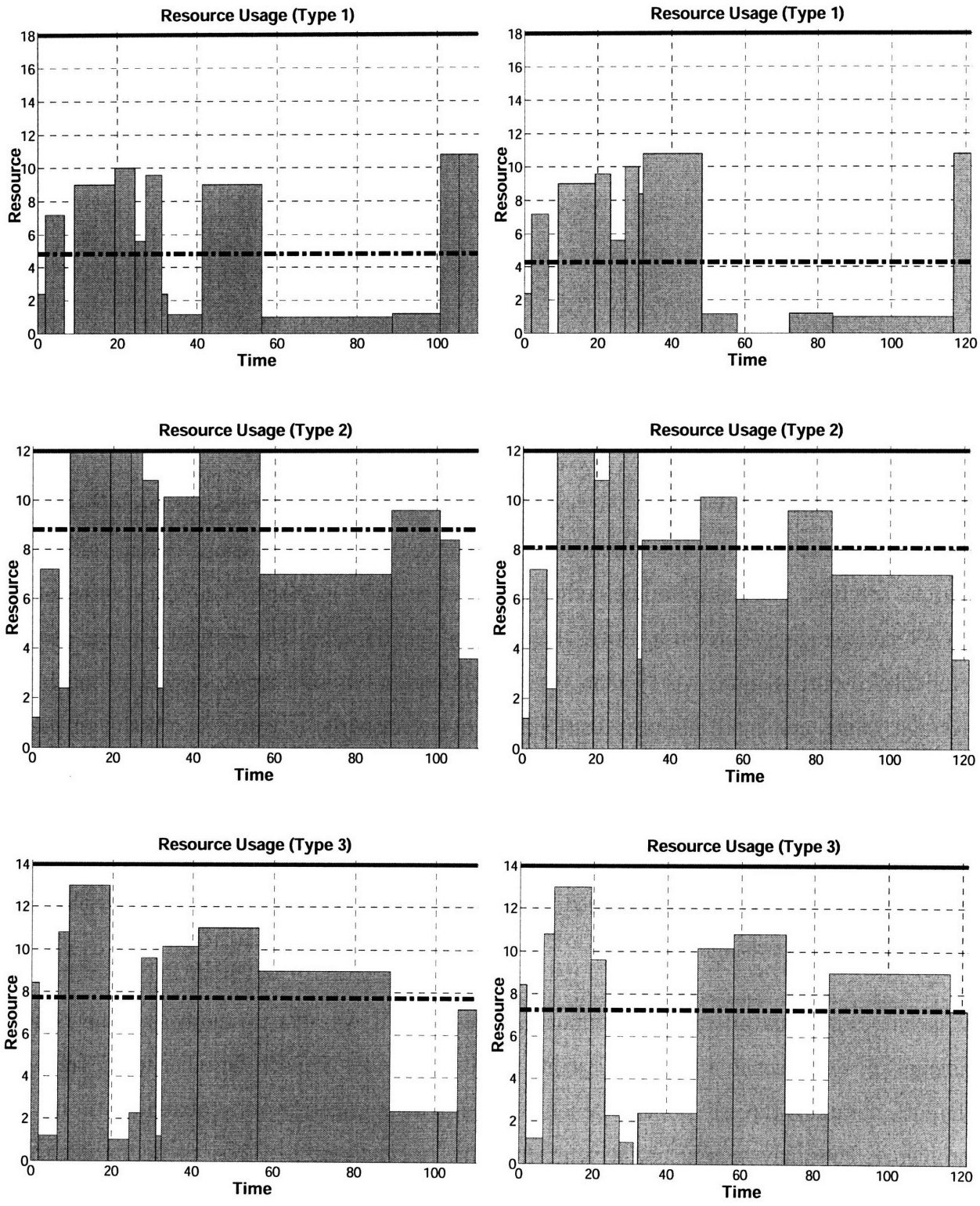


Figure 5-22 Resource usage comparison

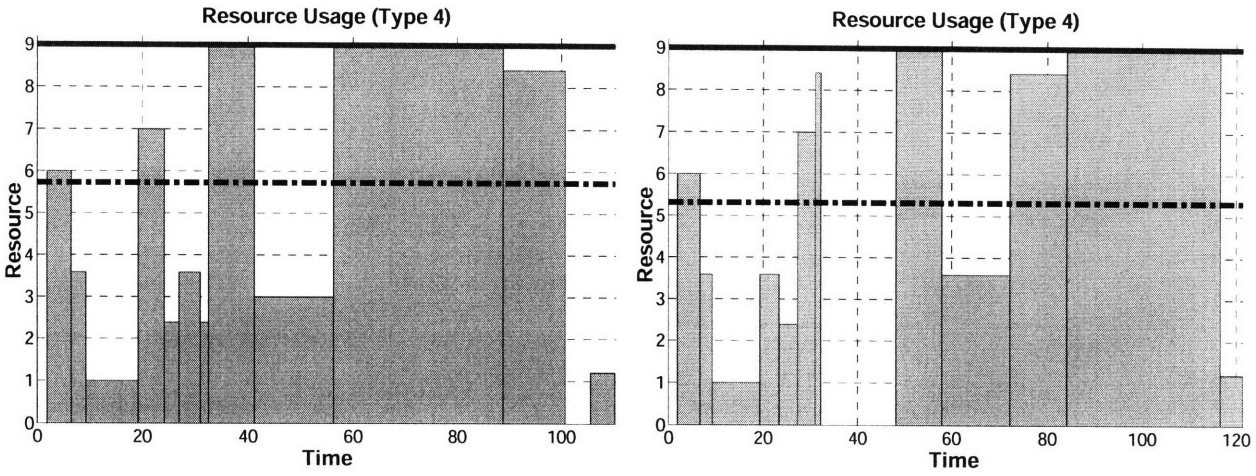


Figure 5-22 Resource usage comparison (cont')

Figure 5-22 compares the resource usage for the two solutions. The plots on the left show the resource usage of the optimal solution. The plots on the right show the resource usage of the sub-optimal solution. The red solid line is the amount of resource available. The black dash-dot line shows the average resource usage over the entire project duration. The optimal solution uses the available resources more wisely than the sub-optimal solution. This is shown by the higher average values and more uniform distributions of the resource usage. It is a result of greater parallelization of tasks. We can see that resource type 1 and resource type 4 are not used for two long periods respectively in the sub-optimal solution, but it never occurs in the optimal solution. A more uniform usage of resource is advantageous since it is easier for the managers to decide how much resource should be allocated to a project. It minimizes the chance of over-allocation and hence reduces wastage.

It is interesting that all the GA solutions suggest that if we allocate more than the nominal amounts of resources to a task, the T mode is preferred to the P mode. In other words, reducing the task durations directly is more beneficial than reducing the rework probabilities in our test problems. This may simply be the artefact of the test data we use. For all our problems we assume that if the T mode is used, the addition (or reduction) of resources only results in reductions (or increases) in the task durations, whereas the quality of work (or the rework probabilities) remains unchanged. Similarly, if the P mode is chosen, only rework probabilities are adjusted, whereas the task durations remain the same. In this particular example, for a 10%

deviation from the nominal resource requirement values, we set the deviation of task durations from their original values to be 5%, whereas the deviation of rework probabilities from their nominal values is set to be 20%. Even though we assume much lower percentage change in task duration than in rework probability (5% vs 20%), the fact that T mode is still preferred tells us that the effect of reducing task durations is much greater and immediate. This is probably also true in real life: it is always easier to finish a job faster than to improve its quality. Realizing that to minimize duration is the objective of our optimization problem, and quality of work (rework probability) is not *directly* related to the duration, it is then not surprising to observe such preference by the GA. If the rework probabilities are extremely high (close to 1), the possible reduction in rework probabilities is big and the possible reduction in task durations is extremely low (close to 0), we will probably observe a shift in the preference. The threshold values for such a shift in preference is left for future studies.

Another possible extension is to assume a mix of T mode and P mode in the model. In this case, the resource levels affect the task durations and the rework probabilities simultaneously, but with different proportions. The trade-off between the task durations and rework probabilities may be governed by some equations. In fact, this assumption makes the model more realistic. However, to realise it more sophisticated governing equations and data are required.

From this example we can see that being able to divide a project into meaningful stages that consist of reasonable numbers of tasks is crucial for project planning. Decisions are then made in stages at important turning points. All these decisions affect the overall success of the project. Moreover, parallelization reduces the total project execution time in general, leaving fewer amounts of resources idle, and therefore should be maximized.

5.4 Chapter Summary

In this chapter, we present the GA solutions to five test problems. These solutions are compared with known optimal solutions to special cases of the problems and solutions obtained using single heuristic rules. The general performance of the GA solutions are satisfactory. Possible improvements to the solution method are outlined. We also examine the GA solution to a small

example in great details. Interpretations to various observations are provided. Some managerial insights are gained through these analysis.

CHAPTER 6
CONCLUSION

In this thesis, we have presented a simulation-based optimization model for solving RCPSPs. We use DSM as our primary data repository and representation tool. It provides clear visualization, eases modifications to RCPSPs, and allows us to use many mathematical analytic methods to analyse a problem. More importantly, rework, which commonly occur in process development projects, can now be easily modelled through the use of rework probabilities and rework impacts using DSM.

Apart from the normal assumptions made in general RCPSPs, we introduce stochastic rework into our model. Most papers in existing literature convert RCPSPs with rework to simpler deterministic problems before they search for a solution. Even though quick solutions can be found to these easier problems, the exact effect of rework on the projects is simply avoided and left unexplored. Our simulation-based approach allows us to generate rework according to rework probabilities and rework impacts stochastically. Moreover, instead of a single value of total project durations, we examine a distribution of total project durations, as a result of many Monte Carlo simulation runs. This way, we can see clearly how rework evolves and contributes to the total project durations.

Moreover, resource constraints are allowed to vary within given ranges in our model. Two work modes that correspond to the resource variations are used. Whenever the amounts of resources allocated to a task differ from the nominal resource requirement values, one can choose whether to change the task durations accordingly (to use T mode) while keeping rework probabilities fixed or vice versa (to use P mode).

To deal with resource conflicts in the resource constrained projects, we use a ranking scheme that is based on a weighted sum of several heuristics. Tasks that are ranked higher have higher priorities to get the necessary resources.

We have coded a GA with special features that are catered to solve our own RCPSPs. The GA is used to find the optimal weights and work modes for several numerical examples. Since the

exact optimal solutions to these numerical examples are unknown, the performance of the solutions to the problems are assessed based on comparisons with solutions that are obtained using other methods and known solutions to special cases of our RCPSPs.

The GA solutions are better than the solutions obtained using single heuristic rules. Our GA also performs much better than another GA in the literature on special cases of our RCPSPs.

In our numerical examples, the T mode seems to be preferred to the P mode when resource levels are high. In other words, using extra resources to reduce the task durations alone results in greater reduction in total project duration than using extra resources to reduce the rework probabilities alone. We expect a shift in this preference, when we increase the rework probabilities and their variation bounds, while reducing the task duration variation bounds. However, the exact threshold for the shift to be observed is still unknown.

From our analysis of the numerical examples, we also see that better solutions always result in greater extent of parallelization of tasks. This leads to higher resource utilization as well as more uniform use of resources over the project span. Moreover, to facilitate decision making it is important for us to be able to identify and divide the tasks into right groups of reasonable sizes. The start of every group of tasks represents a critical point in the project span. Decisions made regarding task sequences and resource allocations at these critical points will have serious implications on the success of the entire project.

At present, the amount of research done related to resource constraints in the DSM literature is very limited. There is an urgent need for realistic test data. Even though there has always been a desire to study the effect of resources on task durations and rework probabilities, no relevant data is available in the DSM literature. It would be interesting to see our proposed model being used to test some real problems with more realistic resource-related data.

In our model, we assume piecewise linear relations about the resource levels with task durations, and resource levels with rework probabilities. However, these piecewise linear relations can virtually be replaced with any reasonable relations, provided more sophisticated data is available. Moreover, instead of assuming that the T mode and the P mode are mutually exclusive, a mix of the two can also be used in the model.

We have shown that our approach of weighted sum of several heuristic rules works better than using single heuristic rules. To explore this approach further, rather than limiting the ranking factors to the present four, other project related information can also be added as ranking factors. We believe that the more complete the information is captured by the ranking factors, the better the GA solutions will be.

REFERENCES

[1] Abdelsalam, H. M. E., and Bao, H. P., *A simulation-based optimization framework for product development cycle time reduction. IEEE Transactions on Engineering Management,* Vol. 53, No. 1, February, 2006, pp. 69-85.

[2] Azadivar, F., *Simulation optimization methodologies. Proceedings of the 1999 Winter Simulation Conference,* pp. 93-100.

[3] Browning, T. R., *Modeling and analysing cost, schedule, and performance in complex system product development,* Ph.D. thesis, Massachusetts Institute of Technology, December, 1998.

[4] Browning, T. R., and Eppinger, S. D., *Modeling impacts of process architecture on cost and schedule risk in product development. IEEE Transactions on Engineering Management,* Vol. 49, No. 4, November, 2002, pp. 428-442.

[5] Browning, T. R., and Yassine, A. A., *A random generator of resource-constrained multi-project network problems,* TCU Neeley School of Business, 2008, Working Paper.

[6] Browning, T. R., and Yassine, A. A., *Resource-constrained multi-project scheduling: priority rule performance revisited,* TCU Neeley School of Business, 2008, Working Paper.

[7] Cho, S. H., and Eppinger, S. D., *A simulation-based process model for managing complex design projects. IEEE Transactions on Engineering Management,* Vol. 52, No. 3, August, 2005, pp. 316-328.

[8] Coley, D. A., *An introduction to genetic algorithms for scientists and engineers,* World Scientific, 1999.

[9] Corby, R. J., *Using the design structure matrix and systems thinking to develop a requirements driven automotive closures design process,* Master thesis, Massachusetts Institute of Technology, February, 2008.

[10] Demeulemeester, E., and Herroelen, W., *A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. Management Science*, Vol. 38, No. 12, December, 1992, pp. 1803-1818.

[11] Demeulemeester, E. L., Herroelen, W. S., *New benchmark results for the resource-constrained project scheduling problem. Management Science*, Vol. 43, No. 11, November, 1997, pp. 1485-1492.

[12] Dong, Q., Representing information flow and knowledge management in product design using design structure matrix, Master thesis, Massachusetts Institute of Technology, December, 1998.

[13] *GA FAQ*, URL: http://www.geneticprogramming.com/ga/GAfaq.html [cited on 02 August, 2008].

[14] Holmes, P., *What are gray codes*, URL: http://cafaq.com/extra/gray.html [cited 02 August, 2008]

[15] Kolisch, R., and Sprecher, A., *PSPLIB – a project scheduling problem library. European Journal of Operational Research*, Vol. 96, 1996, pp. 205-216.

[16] Kolisch, R., *Library for project scheduling problems*, URL: http://129.187.106.231/psplib/ [cited 02 August, 2008]

[17] McGill, E. A., *Optimizing the closures development process using the design structure matrix*, Master thesis, Massachusetts Institute of Technology, June, 2005.

[18] Noor, M. J., *A comprehensive approach to complex system product development: operations management tools applied to automotive design*, Master thesis, Massachusetts Institute of Technology, May 2007.

[19] Pet-Edwards, J., and Mollaghasemi, *Application of genetic algorithm in resource constrained network optimization. Proceedings of IEEE International Conference on Intelligent Systems for the 21st Century*, pp. 3059- 3062, 22-25 October, 1995.

[20] Qian, X., and Tang, D., *Research on resource optimization of concurrent product development process on DSM. Proceedings of 2007 IEEE International Conference on Grey Systems and Intelligent Services*, pp. 1538-1543, Nanjing, China, November 18-20, 2007.

[21] *The Design Structure Matrix Web Site*, URL: http://www.dsmweb.org/ [cited 02 August, 2008]

[22] *The Mathematics of Scheduling: Directed Graphs and Critical Paths*, URL: http://www.ctl.ua.edu/math103/scheduling/schedmnu.htm#Scheduling%20Algorithms [cited 02 August, 2008].

[23] Thomas, P. R., and Salhi, S., *A Tabu search approach for the resource constrained project scheduling problem. Journal of Heuristics*, Vol. 4, 1998, pp. 1485-1492.

[24] Wall, M. B., *A genetic algorithm for resource-constrained scheduling*, Ph.D. thesis, Massachusetts Institute of Technology, June, 1996.

[25] *Wikipedia: Monte Carlo Method*, URL: http://en.wikipedia.org/wiki/Monte_Carlo_method [cited 02 August, 2008].

[26] *Wikipedia: Triangular Distribution*, URL: http://en.wikipedia.org/wiki/Triangular_distribution [cited 02 August, 2008].

[27] Zambito, A. P., *Using the design structure matrix to streamline automotive hood system development*, Master thesis, Massachusetts Institute of Technology, January, 2000.

APPENDIX

PROGRAM DOCUMENTATION

```
Main (script file) - Main Program for GA
Called by:  GA
Calls:
```

- SetPara (script file) - set fixed parameters
- InitialPopulation - initialize Gray strings
- FindUnknowns - convert Gray strings to real numbers
- FindFitness (script file) - compute fitness of each individual
- Statistics - compute sum of fitness, mean fitness, max fitness and fittest individual
- Scaling - scale the fitness values
- Selection - select one individual each time based on fitness proportion
- CrossOver - perform crossover to get 2 new individuals for next generation
- NoCrossover - selected individuals go to the next generation without crossover
- Mutate - randomly alter selected bits to preserve biodiversity
- Replace - replace the old population with the new one

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

SetPara (script file) - set fixed parameters for Monte Carlo simulation and
GA
Called by: Main
Calls: none

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

InitialPopulation - randomly initialize Gray strings
Called by: Main
Calls: none
In: structure s
Out: structure s with population initialized

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FindUnknowns - convert Gray strings to real numbers and normalize the weights
Called by: Main
Calls: FindIntegers_gray - convert Gray strings to integers within the pre-
defined ranges
       Num2Gray - convert normalized weights to Gray strings
In: structure s
```

```
Out: structure s with Gray strings converted to real-valued parameters
(s.Vars)

.....


FindIntegers_gray - convert Gray strings to integers within the pre defined
ranges
Called by: FindUnknowns, if Gray strings are used
Calls: none
In: structure s
Out: structure s with Gray strings converted to integers (s.Itgs)

.....

Num2Gray - convert normalized weights to Gray strings
Called by: FindUnknowns
Calls: none
In: structure s
Out: structure s, with strings representing normalized weights

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FindFitness (script file) - compute the fitness values
Called by: Main
Calls: TriPDF, CriticalPath, CalRank, NewP, NewT
*Eric McGill first coded Browning's algorithm in MATLAB [17].

.....

TriPDF - update task durations when resource levels differ from the nominal
values [17]
Called by: FindFitness
Calls: none
In: BCV, MLV, WCV of a triangular distribution
Out: a random value from the distribution
Author: Eric McGill

.....

CriticalPath - compute the critical time of the tasks, the critical and sub-
critical paths
Called by: FindFitness
Calls: none
In: M, lower triangular part of DSM first plane
Out: critical times, critical path, and sub-critical paths
```

```
.....................................................................

CalRank - calculate the ranking factors of a task
Called by: FindFitness
Calls: none
In: task number, network data, WorkThen, MayWorkNow, durpenalty, CT
Out: ranking factors

.....................................................................

NewP - update rework probabilities when resource levels differ from the
nominal values
Called by: FindFitness
Calls: none
In: DSM, task number, nominal resource requirement value, actual resource
level, resource and rework probability variation factors
Out: DSM, with new rework probabilities

.....................................................................

NewT - update task durations when resource levels differ from the nominal
values
Called by: FindFitness
Calls: none
In: DSM, task number, nominal resource requirement value, actual resource
level, resource and task duration variation factors
Out: new task duration

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Statistics - compute sum of fitness, mean fitness, max fitness and fittest
individual
Called by: Main
Calls: Elite (if Elitism is on) - randomly replace an individual with the
elite member of the previous generation if this elite member is fitter than
the current fittest individual
In: s
Out: s

.....................................................................

Elite - randomly replace an individual with the elite member of the previous
generation if this elite member is fitter than the current fittest individual
Called by: Statistics, if elism is on (s.Elit = 1)
Calls: none
In: s
Out: s
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Converge - check whether stopping criterion is met
         - this function is currently disabled
Called by: Main
Calls: none
In: structure s
Out: done (0 or 1), s

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Scaling - scale the fitness values
        - to enable this function, s.SConst!=0 and s.ScaleStart<1
Called by: Main
Calls: none

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Selection
  - select one individual each time based on fitness proportion
  - relative fitness is used, i.e. actual fitness - min.fitness
Called by: Main
Calls: none
In: s, parent index (1 or 2)
Out: s

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

CrossOver
  - perform crossover to get 2 new individuals for next generation
  - swap the bits to the right of CrossSite
Called by: Main
Calls: none
In: structure s
Out: structure s

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

NoCrossover - selected individuals go to the next generation without
crossover
Called by:  Main
Calls:      none
In:         structure s
Out:        structure s

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
Mutate - randomly alter selected bits to preserve biodiversity
Called by: Main
Calls: none
In: structure s
Out: structure s

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Replace - replace the old population with the new one
Called by: Main
Calls: none
In: structure s
Out: structure s
```