uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Nader Azizi

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

Ph.D. (Mechanical Engineering)

GRADE / DEGREE

Department of Mechanical Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Manufacturing Productivity Improvement:
A Study of Human Boredom, Job Rotation and Scheduling

TITRE DE LA THÈSE / TITLE OF THESIS

Ming Liang

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

Saeed Zolfaghari

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Balbir Dhillon

Benoît Montreuil

Xiao Huang

Dan-Sorin Necsulescu

Gary W. Slater

Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

Manufacturing Productivity Improvement: A Study of Human Boredom, Job Rotation and Scheduling

Nader Azizi

A thesis submitted to the Faculty of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

In Mechanical Engineering
Ottawa-Carleton Institute for Mechanical and Aerospace Engineering
University of Ottawa
Ottawa, Ontario
Canada

April 2009

Your file Votre référence
ISBN: 978-0-494-51789-5
Our file Notre référence
ISBN: 978-0-494-51789-5

Abstract

The focus of this thesis is the development of methodologies for the analysis of two important issues in a manufacturing environment: human boredom and machine utilization. In particular, this study proposes a mathematical programming model for job rotation to cope with worker's boredom in a manufacturing cell. One of the important side effects of task rotation is the impact of rotation interval on workers' skill learning and forgetting which may cause productivity losses. For this reason, the proposed model simultaneously incorporates worker's skill and boredom variations. The developed formulation is a mixed integer nonlinear mathematical programming model. A linear version of the model is also presented by assuming that both worker's skill and boredom change linearly over time. Furthermore, given the complexity and uncertainty involving human emotions, a novel approach based on the state-of-the-art Bayesian Networks is also presented to measure and to predict human boredom at work. First, the static Bayesian Network is introduced to capture the static aspect of boredom modeling. The static boredom model is subsequently extended based on dynamic Bayesian Networks to account for temporal aspect of boredom modeling. The dynamic boredom model allows integrating boredom evidences spatially and temporally, thus providing a more robust and accurate boredom inference. The proposed boredom model is validated using case based scenarios. Moreover, a metaheuristic approach based on several well-known search algorithms is also presented to solve the nonlinear version of the job rotation model. The proposed algorithm integrates several ingredients including a simulated annealing module, three types of memory, an evolutionary operator, and a blockage removal feature in a generic framework. The main characteristic of the proposed model is the use of both positive and negative memory to intensify the search around good solutions as well as an evolution-based diversification approach. This generic metaheuristic can be tailored to solve a variety of hard optimization problems.

Another important issue in a manufacturing environment is machine utilization which is largely affected by scheduling decisions. Though various scheduling problems have been investigated for several decades, the lack of efficient solution methods is still

hindering the machine utilization and hence the manufacturing productivity. For this reason, the two most common scheduling problems, job shop scheduling and flow shop scheduling problems are investigated in this thesis. The proposed new metaheuristic is used to solve the scheduling problems.

This study presents a methodology to explicitly deal with human's boredom and skill variations at work. It also introduces a probabilistic model to quantitatively measure and predict human boredom. Finally, it contributes to the development of computational intelligence by introducing a generic framework of a new metaheuristic to solve a class of combinatorial optimization problems.

Acknowledgments

Over the past four years, I have had the privilege to work with two extraordinary and outstanding educators, Professor Ming Liang (University of Ottawa) and Professor Saeed Zolfaghari (Ryerson University). I would like to express my sincere gratitude to my supervisors, Dr. Zolfaghari and Dr. Liang, for their encouragements and unconditional supports throughout my studies at the University of Ottawa. I am grateful to my thesis committee members, Dr. Benoît Montreuil (Laval University), Dr. Balbir Dhillon, Dr. Xiao Huang (Carleton University), and Dr. Dan-Sorin Necsulescu for their valuable and constructive comments. I also wish to thank faculty members, staffs, and my colleagues at the Department of Mechanical Engineering for their friendship and help during my student life at the University of Ottawa. Last but not least, special thank to my family in Iran for their love, encouragement, and support.

Contents

List of Figures

List of Tables

Nomenclature

Indices:

i	index of workers, $i = 1,2,3,...,m$
j	index of operations (workstations), $j = 1,2,3,...,m$
t	index of time in production horizon, $t = 1,2,3,...,T^H$
k	number of time(s) worker i is assigned to workstation j during production horizon T, $k = 1,2,3,...,n$
e	index of jobs, $e = 1,2,3,...,n^e$
l	index of machines, $l = 1,2,3,...,m^l$

Sets:

I	set of workers operating in the production cell $\mathbf{I} = \{1,2,3,...,i,...,m\}$
J	set of operations to be performed (or set of workstations) $\mathbf{J} = \{1,2,3,...,j,...,m\}$
T	set of time units in production horizon $\mathbf{T} = \{1,2,3,...,t,...,T^H\}$

Variables:

$x_{i,j,t}$	1, if worker i performs operation j at time t; 0, otherwise
$S_{i,j,t}$	skill level of worker i in performing operation j at time t
$S_{i,j,t}^{\mathrm{Re}m}$	remnant of skill j at time t
$V_{i,j,t}$	motivation level of worker i in performing operation j at time t
$t_{i,j}^{A}$	time at which the skill of worker i in performing operation j reaches the upper bound
$t_{i,j}^{A'}$	time at which the remnant of skill j gained by worker i reaches the lower bound
$a_{i,j}^{k}$	time at which worker i begins (or has begun) operating at workstation j
$t_{i,j}^{C}$	time at which the motivation level of worker i in performing operation j reaches the upper bound

$t^{C'}_{i,j}$ time at which the motivation level of worker i in performing operation j reaches the lower bound

$d^{L}{}_{i,j}$ last departure time of worker i from workstation j

$t^{mdc}_{i,j}$ time at which the motivation of worker i in performing operation j begins declining

$D_i^{V^{UB}}$ minimum time required for the motivation of worker i to raise from an initial level, $V^{Initial}$, to the upper bound

$v^1_{i,j}$ 1, if $t \geq t^A_{i,j}$; 0, otherwise

$v^2_{i,j}$ 1, if $t \geq t^{A'}_{i,j}$; 0, otherwise

$v^3_{i,j}$ 1, if $t < t^{A'}_{i,j}$; 0, otherwise

$v^4_{i,j}$ 1, if $d^L_{i,j} = 1$; 0, otherwise

$u^1_{i,j}$ 1, if $t^C_{i,j} \leq t < t^{mdc}_{i,j}$; 0, otherwise

$u^2_{i,j}$ 1, if $t < t^C_{i,j}$; 0, otherwise

$u^3_{i,j}$ 1, if $t \geq t^{C'}_{i,j}$; 0, otherwise

$u^4_{i,j}$ 1, if $t \geq t^{mdc}_{i,j}$;0, otherwise

θ_l temperature at iteration l

q number of neighbours on a critical path

S_{ms} size of the short-term memory

L_{ms} size of the long-term memory

Parameters:

T^H production horizon

$S^{Initial}$ initial level of skills

S^{UB} skill upper bound

S^{LB} skill lower bound

$S^{\max}$ theoretical maximum level of skills

$S^{\min}$ theoretical minimum level of skills

$V^{Initial}$ initial level of motivation

V^{UB} motivation upper bound

V^{LB} motivation lower bound

$V^{\max}$ theoretical maximum level of motivation

$V^{\min}$ theoretical minimum level of motivation

α parameter that determines the relative importance of each criterion in objective function

β_i learning slope of worker i

r_i learning curve coefficient associated with worker i

γ_i forgetting slope of worker i

f_i forgetting curve coefficient associated with worker i

τ_i motivation (boredom recovery) slope of worker i

θ_i boredom (motivation reduction) slope of worker i

h_i motivation coefficient of worker i

g_i boredom coefficient of worker i

ε_1 upper bound skill threshold

ε_2 lower bound skill threshold

δ_1 motivation upper bound threshold

δ_2 motivation lower bound threshold

$\Omega_{i,j}$ minimum time interval between the two consecutive assignments of worker i to operation j

M_l sufficiently large numbers

N number of jobs

M number of machines

P_{ij} processing time of job j on machine i

PopSize population size

L iteration number

θ_1 initial temperature

λ control parameter

γ coefficient of the number of unacceptable moves

TabuSize length of the tabu list

CT complement of the temperature

Ω coefficient of diversification

Q number of the selected genes on a chromosome

Chapter 1

Introduction

The manufacturing productivity is affected by both the human and machine factors. Much of the previous research has been focused on the machine aspects due to the difficulties in modeling the human issues. Nevertheless, the human aspects in a manufacturing facility not only affect the productivity but also influence safety and machine utilization. Therefore, such issues have to be carefully addressed.

In the literature, some researchers have considered the workforce skill related issues in the worker task assignment problem. Warner *et al.* (1997) suggested assigning workers to machine cells based on their human and technological skills. Warner *et al.* (1997) described technological skills as mechanical, mathematical and measurement ability while human skills referred to as communication skills, leadership, teamwork, and decision-making ability. Bhaskar and Srinivasan (1997) presented mathematical programming models for static and dynamic worker assignment problems in cellular manufacturing systems. The static case refer to the situation in which the same allocation is used for processing all product varieties, while in dynamic case operators are allowed to move from one station to another in producing different varieties. Molleman and Slomp (1999) presented a linear goal programming method to assign the workers to a specific task in order to minimize shortages, makespan, and production time.

A number of researchers have investigated the impact of training on the performance of manufacturing systems to establish appropriate training programs. Askin and Huang (1997) proposed a solution procedure based on an integer programming model for assigning workers to cells and selecting a specific training program for each worker. Norman *et al.* (2002) developed a team based model for worker assignment in manufacturing cells that considers both technical and human skills. Norman *et al.* (2002) showed that if human skills are explicitly taken into account in both worker training plan and assignment strategies, the performance of the cell will improve significantly. Cesani

and Steudel (2000b) conducted an empirical investigation in a two-operator cell and found that when more than one operator is responsible for a machine, the system performance improves significantly. Kher (1999) studied worker training related issues with the simultaneous presence of learning, forgetting and worker attrition effects. Kher (2000) and Kher *et al.* (1999) conclude that the effectiveness of cross-training depends significantly on the existing forgetting rate of the workers.

Numerous surveys have found that there is significant and positive correlation between repetitive task and boredom. Unlike worker assignment and training issues, worker boredom has not been explicitly addressed in the accessible literature. Boredom is a common and sometimes very serious problem in real-life situations. It can be associated with performance reduction, general dissatisfaction, and accidents. The factors that cause boredom are complex but almost always include exposure to repetitive stimulation. Therefore, job rotation could be a potential solution to the worker's boredom or lack of motivation (Bhadury and Radovilsky , 2006, Ference *et al.*, 1977).

Job rotation is defined as lateral transfer of workers among a number of different workstations where each requires different skills and responsibilities. In addition to its effect on worker's boredom, a number of studies have addressed several other benefits of job rotation to workers and firms. As a benefit to firms, job rotation is said to improve a firm's ability to deal with change. Flexible workers can provide buffer against uncertainties in manufacturing systems (Anselmi and Sundarrajan, 2000). As a benefit to workers, job rotation may increase worker job satisfaction (Cunningham and Eberle, 1990, Davis and Taylor, 1989). Job rotation also yields such benefits to workers as reducing the injuries due to performing repetitive tasks as well as the worker's fatigue especially if the worker is exposed to various muscular loads during task operation (Hinnen *et al.*, 1992, Henderson, 1992).

One of the costs associated with job rotation is the effect of rotation intervals on worker learning and forgetting which may cause productivity losses. Once a worker is transferred into another workstation, he (she) starts learning a new set of skills while his (her) previous skills begin diminishing. Productivity losses during learning at the new workstation are one of the costs associated with workers rotation. If the worker assignment to the other workstation is too long he or she may lose his or her entire skill

set. In such cases, the cost of training or retraining should be also considered. On the other hand, if the job rotation is done too frequently, the worker may have to spend much of his/her time to adapt to new tasks and may never be able to utilize his/her capability to the full extent. Therefore there is a trade-off between the benefits of long and short rotation intervals. However, when many jobs of different requirements and workers of different capabilities are involved, the job rotation problem becomes very complex. This calls for a scientific modeling and solution approach.

Another important issue in a manufacturing environment is machine utilization which is largely affected by scheduling decisions. Though various scheduling problems have been investigated for several decades, the lack of efficient solution methods still is hindering the machine utilization and hence the manufacturing productivity. For this reason, the two most common scheduling problems, job shop scheduling and flow shop scheduling problems will be investigated in this thesis. A new hybrid solution method will be developed in a generic metaheuristic format. This generic metaheuristic can be tailored to solve a variety of hard optimization problems. In this thesis work, it will be used to solve the job rotation and scheduling problems.

1.1 Thesis outline

The results of many studies confirm that job boredom could negatively impact productivity, safety and human health particularly in a repetitive manufacturing environment. Job rotation has been recommended by many researchers as a tool to cope with employee's boredom at work. Although job rotation may reduce the employee's boredom, a poorly designed job rotation plan may conversely affect worker skill improvement and hence productivity. Therefore, an ideal rotation plan to cope with employee's boredom should also account for worker's skill variation.

According to accessible literature, a comprehensive study on the methodologies and the implementation issues of such job rotation scheme is yet to be conducted.

On the other hand, many studies have been dedicated to improve manufacturing productivity through enhancing machine utilization in the past several decades. Machine utilization is mostly affected by scheduling decisions. Despite all efforts and advances in

computing technology, the manufacturing productivity is still suffering from the lack of efficient solution technique for scheduling problems.

The questions this study attempts to answer include:

a) How boredom at work could be measured and possibly predicted?
b) How workers in a manufacturing environment should be rotated to reduce job boredom while the effect of rotation on their skill is minimal?
c) If such job rotation is modeled, which workers should be included in the rotation plan? What is the rotation sequence and what is the correct rotation interval length?
d) How scheduling problems could be solved more efficiently to improve manufacturing productivity?

The above questions could be answered by concentrating on the following topics:

1) Developing a methodology to quantitatively measure and predict human boredom at work.
2) Developing a mathematical model for job rotation that incorporates both worker's boredom and skill variations.
3) Developing a solution technique to solve the job rotation problem.
4) Developing efficient solution techniques for the two popular scheduling problems, job shop and flow shop.

Boredom at work is measured by self reporting scales. No quantitative measure has been developed and reported for this type of emotion in the accessible literature. To address the first issue, a probabilistic model based on Bayesian Networks is developed in this research to quantitatively measure and predict human boredom at work. Both static and dynamic Bayesian Networks will be introduced to deal with static and temporal aspects of human boredom modeling.

The previous studies on job rotation have been primarily concentrated on minimizing the negative impact of work itself (e.g., low back injuries) as well as the working environment (e.g., exposure to noise) on employees.

To address the second issue, in this study a mathematical programming model for job rotation that *simultaneously* incorporates employee's boredom and skill variations

will be developed. In the proposed model, the skill variations are formulated by generalizing the concepts of learning and forgetting phenomena. Furthermore, to model the employee's boredom, it is assumed that like many other natural phenomena, boredom may grow or decay exponentially.

The developed job rotation model is a nonlinear mixed integer programming model. To address the third issue, a generic framework of a new metaheuristic that integrates several well known search algorithms: simulated annealing, tabu search, and genetic algorithms will be presented to solve the job rotation model. The main components of the proposed metaheuristic are a simulated annealing module, three types of memory, and an evolutionary operator.

Finally, several hybrid algorithms based on the generic framework of the new metaheuristic are tailored to solve the two well known NP-hard, job shop and flow shop scheduling problems.

The overall structure of this thesis is summarized in Figure 1.1.

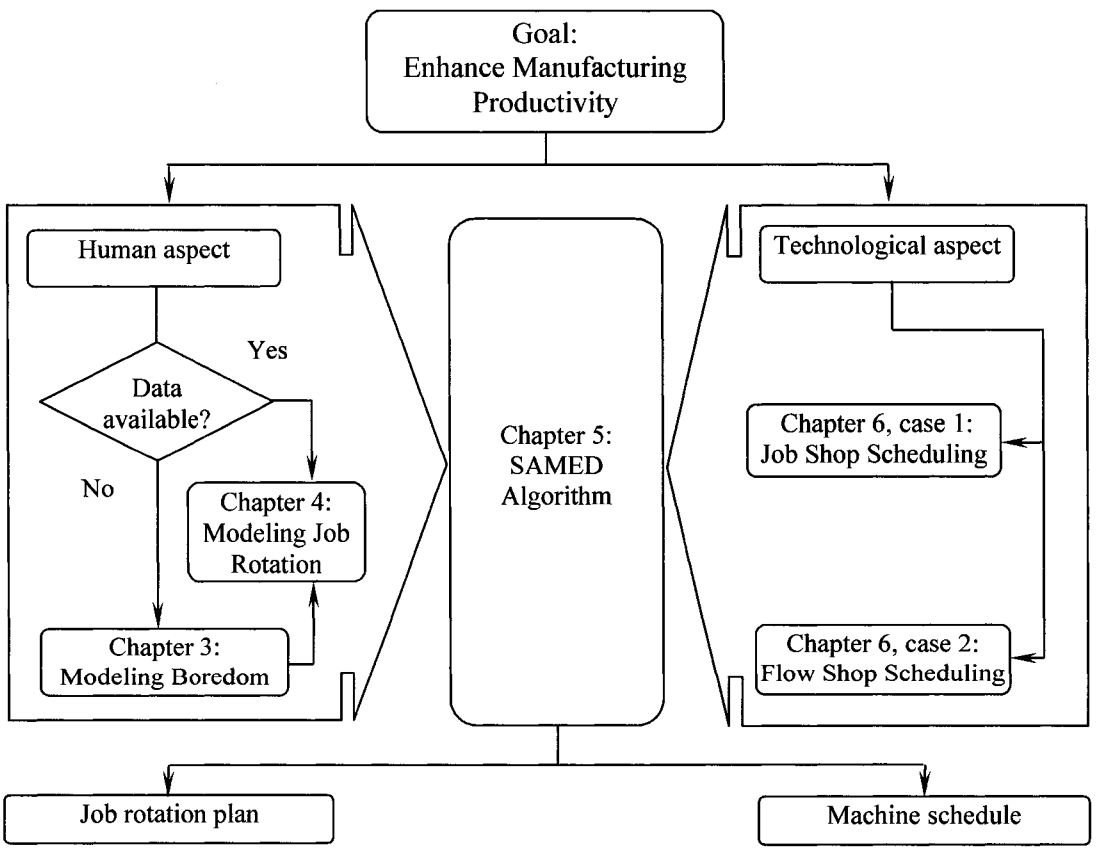


Figure 1-1. Structure of the thesis

Chapter 2

Literature Review

In this chapter, a brief review of the literature related to job rotation, boredom modeling, job scheduling, and solution methods is presented.

2.1 Review of job rotation modeling

A number of studies have modeled job rotation with the objective of reducing the physical impact of work and workplace on employees. Carnahan *et al.* (2000) studied several methods to integrate safety criteria into scheduling algorithms to produce job rotation schedules that reduce the potential for injury. In the work by Carnahan *et al.* (2000), low-back injury was the safety concern with criteria developed from the Job Severity Index (JSI). Tharmmaphornphilas *et al.* (2003) proposed a mathematical model for job rotation. The model minimizes daily noise exposure encountered among workers in a sawmill.

2.2 Review of boredom modeling

Boredom has attracted attention from researchers in diverse disciplines including human factors engineering, psychology, sociology, education, criminology, and industrial psychology. However, it has not been studied extensively and there is no coherent, universally accepted definition for boredom (Vodanovich, 2003). O'Hanlon (1981) defined boredom as a "unique psychophysical state that is somehow produced by prolonged exposure to monotonous stimulation". Hill and Perkins (1985) stated that "boredom occurs when stimuli is construed as subjectively monotonous". Mikulas and Vodanovich (1993) defined boredom as "a state of relatively low arousal and dissatisfaction which is attributed to an inadequately stimulating environment". Fisher (1993) further defined boredom as "an unpleasant transient state in which individuals feel

an extreme lack of interest in their current activity". Probably, the lack of an agreed definition of boredom could be partly accounted for the existence of various approaches to measure different subsets of boredom such as free time boredom, boredom susceptibility, and job boredom. In the term "free time boredom", free time is defined as time remaining after daily obligations are met (Ragheb and Merydith, 2001).

2.2.1 Boredom causes and effects

There is no comprehensive theory for boredom causes. However, enough evidences have been provided to show it has serious consequences (Fisher, 1993). The outcomes of boredom are considered to be largely negative. It has been associated with absenteeism, job dissatisfaction, accidents, reduced performance on vigilance tasks, and performance variability (Drory, 1982).

Early research assumed that boredom causes are outside the person, such as repetitive tasks. However, recent studies show that there are other likely causes of boredom. The main factors causing boredom at work could be classified as work condition, environmental condition, and person effect.

Several work-related causes of boredom that have been reported in literature are: qualitative under load, quantitative under load, and qualitative overload work. Types of work characterized as qualitative under load are likely experienced as boring by most people. The results of numerous surveys show that people often reported being bored when they do repetitive and simple tasks. Once learned, simple or repetitive tasks require very little mindful attention. Examples of such tasks are vigilance, inspection, checking, and driving. Some jobs do not contain enough tasks to keep employees occupied for the time they are required to remain at work. In quantitative under load category, individuals working in such condition usually complain about boredom because "they have nothing to do" especially after a busy period. A number of studies (e.g., Fisher, 1993; Locke and Latham, 1990) noted that in a qualitative overload, respondents reported being bored because their jobs were too complicated, requiring high level of skill and continues attention.

The second main cause of boredom is the environmental condition. Two aspects of work environment that may impact employee's boredom are the presence of other people

and organizational control practice. The results for the impact of other people presence on boredom are not consistent. While some studies show that the presence of other people may increase the performance in simple task (Bond and Titus, 1983), other researchers found that in some situations it may, in fact, increase boredom. In Fisher's study for instance, some respondents reported being bored because of unfriendly and uncommunicative coworkers (Fisher, 1987). Another aspect of work environment that may contribute to employee's boredom is the level of organizational control practices. Research shows that workers in firms that prohibit talking or limit breaks are more likely to experience boredom because of the lack of variety in work environment (Ryan 1982, Guest *et al.,* 1978).

The third main cause of boredom is the person effects. Person related effects on boredom are personality, capacity, and mental health. The role of personality in reaction to monotonous task has been studied by a number of researchers. For instance, Gardner and Cumming (1988) suggest that individuals with extroversion personality are more likely to be bored when performing monotonous task than introversions. Early research on the effect of individual differences to boredom suggests that younger, male, extrovert, and intelligent workers are more susceptible to experience boredom (Drory, 1982). Drory (1982) studied the relationship between boredom at work, personal characteristics and performance. The results of Drory's experiment show that all of the variables defining individual capacity including age, education, and experience were positively related to the self reported boredom except for age that exhibits a negative correlation with boredom. Several studies have shown that the cause of repeatedly feelings of boredom might be pathological. Individuals with high scores on measures of Boredom Susceptibility (Zuckerman, 1971, 1979) and Boredom Proneness (Farmer and Sundberg, 1986) tend to experience greater boredom across a range of work situations (Ahmed, 1990). Farmer and Sundberg (1986) describe the boredom prone individual as "one who experiences varying degrees of depression, hopelessness, loneliness, and distractibility".

2.2.2 Measuring boredom

Boredom at work has commonly been measured by self reporting scales. Two Job Boredom Scales have been proposed in the literature. The first scale was introduced by

Grubb (1975). It includes 11 items arranged on a five point scale (e.g., How monotonous is your job?). The response ranges from *hardly at all* (1) to *greatly* (5). The second job boredom scale is suggested by Lee (1986). Lee's job boredom scale consists of 17 items organized also on a five point scale. In Lee's scale, the person is asked to answer (with yes or no) such questions as: Do you often get bored with your work? Do you often get tired on the job? Do you become irritable on the job? A complete list of 17 items of Lee's scale could be found in Appendix A.

2.3 Review of Bayesian Networks

A Bayesian Network (BN) is a state-of-the-art knowledge representation scheme dealing with uncertain knowledge. Also known as graphical models, BNs are probabilistic models based on *Directed Acyclic Graphs*. They deal with two problems of uncertainty and complexity by providing a compact representation of joint probability distributions using a combination of graph theory and probability theory. The graph structure specifies statistical dependencies among the variables while the local probabilistic models specify how these variables are combined. A number of classical probabilistic models that are used widely in such fields as systems engineering, information theory, and pattern recognition are known to be special cases of the general BNs. Theses models include Markov models, Kalman filters, and Ising models (Ji *et al.*, 2006).

Since the introduction of the Bayesian Networks in the late 1970s a number of studies have been conducted, and many networks have been constructed based on this paradigm in a variety of application areas such as industrial applications, military, and medical diagnosis. Bayesian Networks have a number of advantages over other modeling techniques. BN models are robust tools both for graphically representing the relationships between variables and for dealing with uncertainties. The graphical structure of BNs facilitates establishing relationships among variables visually. Furthermore, in Bayesian Networks it is possible to propagate beliefs in a proper way upon the arrival of evidence(s) in order to assess the impact of the evidences on a hypothesis of interest.

Two different approaches have been used to construct Bayesian Networks: data-based approach and knowledge-based approach (Nadkarnia and Shenoyb, 2004). The data-based approach uses conditional independence semantics of BNs to induce models

from data. The knowledge-based approach employs causal knowledge of domain experts in constructing Bayesian networks.

BNs could be classified into Static Bayesian Networks (SBNs) and Dynamic Bayesian Networks (DBNs). SBNs are limited to modeling static events whereas DBNs can be used to model both static and dynamic events.

One of the crucial tasks in modeling a complex problem with BNs is to specify the conditional probabilities. If data are available, special algorithms can be used to learn prior and conditional probabilities of a Bayesian model. This is not always possible as the data in many cases may not be available or too expensive to collect. Another approach to parameterize a BN is to use the knowledge of domain experts, i.e., *knowledge elicitation.* This approach often requires enormous collaboration between knowledge engineers and subject matter experts, which in most cases is difficult to establish. Furthermore, there is a large body of literature covering the psychological biases encountered during elicitation and use of probability values (Fenton *et al.*, 2007).

One possible way to alleviate this problem is to employ the *qualitative explanation of conditional probability* approach (Wellman 1990, Druzdzel and Henrion 1993). The Bayesian Networks based on the qualitative explanation of conditional probability approach is often called *Qualitative Bayesian Networks*. In QBNs, the degree of influence between two adjacent nodes is first determined using the knowledge of domain expert. The degree of influence could be either Strong (positive or negative), Medium (positive or negative), Weak (positive or negative), null (0) or unknown (?). Then, special types of algorithms such as Qualitative Belief Propagation (Druzdzel and Henrion 1993) are used to propagate evidence using the degrees of influences. Nonetheless this approach still does not map these degrees of influence to actual Conditional Probability Tables (CPTs).

2.4 Review of modern heuristic approaches

Simulated Annealing (SA) (Kirkpatrick *et al.,* 1983), Tabu Search (TS) (Glover, 1986), and Genetic Algorithms (GA) (Holland, 1975) are among the earliest popular heuristics. Simulated annealing is a stochastic search technique that has been developed by considering the analogy between the annealing process of solid materials and the procedure of solving optimization problems. In conventional simulated annealing, the

search begins with an initial solution. Then, a neighbouring solution is generated by a randomized perturbation in the current solution. If the quality of a candidate solution is better than that of the current solution, the current solution will be replaced by the candidate solution. Otherwise, a random number between zero and one is generated and is compared to the value of a transition probability. If the random number is less than the value of the transition probability, then the candidate solution, regardless of whether it is worse than the current one, is accepted. Otherwise, another solution from the neighbourhood of the current solution will be considered. The transition probability is a function of the difference between the "costs" of the candidate and current solutions. The transition probability is also affected by a control parameter called *temperature*. At the beginning of the search the temperature is set to a high value allowing the search to accept even worse solutions. However, as the search continues the temperature declines such that at the end of the search only the transition (move) to better solutions will be approved.

Tabu search is a deterministic search procedure that mimics the human decision making process. Tabu search explores the whole neighbourhood of a solution to find the one with the highest quality. Once the best neighbouring solution is found, the move to that solution will be made and the newly obtained solution will be set as the "current" solution for the subsequent iteration. To avoid cycling, whenever a move takes place, the whole or part of the solution that has been recently visited is added to a short-term memory called *tabu list*. In the standard tabu search, the size of the tabu list is fixed and once a new solution (or part of a solution) is added into the list, the last solution (part of the solution) is deleted from the list.

A genetic algorithm works in a way similar to the evolution process of species reproduction (Holland 1975). Unlike SA and TS that work with a single solution, a GA uses a group of solutions called *population*. In genetic algorithms, the initial population is generated either randomly or by means of constructive heuristics. Each member of the population is represented by a string called *chromosome*. A chromosome is composed of a number of elements known as *Genes*. The search begins with an initial population. A fitness value is assigned to each individual in the population according to a problem-specific objective function. To generate a new population, two chromosomes are selected

randomly or based on a probability in favour of improved fitness. The reproduction phase is carried out via two operators. The first operator, named *crossover*, blends parts of the genetic characteristics of two selected parents to create the genetic makeup of an offspring. *Mutation* is the second operator and is used to spontaneously modify the genetic composition of the offspring. This offspring differs from its parents but inherits their certain characteristics. The above steps are repeated until an entire new population is generated.

In recent years, a number of metaheuristics have been developed. Examples include the Greedy Randomized Adaptive Search Procedure (GRASP) (Feo and Resende, 1995, Aiex *et al.,* 2003), Adaptive Multi Start (Boese 1994), Adaptive Memory Programming (AMP) (Taillard *et al.,* 2001), Ant System (AS) (Dorigo and Gambardella, 1997), and Particle Swarm Optimization (PSO) (Kennedy and Eberhart, 1995).

The ant system has been developed based on the way ants communicate with each other as they search for food. Real ants search around their nest for food in a random fashion. Once an ant finds a source of food, it leaves on its way back to the nest a trail of chemical instance called Pheromone to guide other ants to the source of the food. Shortly after, the path to the closer source of the food is highlighted by a large amount of pheromone trails. Ant system was proposed to solve combinatorial optimization problems. It represents ants by "agents" to construct a provisional solution iteratively. To construct the solutions, ants are guided by artificial pheromone trail and other problem information. Once the solutions are constructed, the ants deposit pheromone on the components that they have used in their solutions. In general, the components that contribute to generating a good solution or that have been used by more ants receive a higher amount of pheromone. It is worth to note that before reinforcing the pheromone trails, the level of all pheromones is weakened to avoid trapping the search. Using different ways to update the pheromone has led to different ant system algorithms (Taillard *et al.,* 2001; Gambardella and Dorigo, 1996; Stutzle and Hoos, 2000). Also, it has been demonstrated that a hybrid ant system with a local search provides better results in solving complex problems (Dorigo and Gambardella, 1997; Stutzle and Hoos, 2000; T'kindt *et al.,* 2002).

Particle swarm optimization is a population-based stochastic search technique. It was first reported by Kennedy and Eberhart in 1995. This algorithm is inspired by social behaviour of bird flocking or fish schooling. Generally, the swarm is modeled by particles in multidimensional space. Particles fly through the problem space by following the current optimum particles. They have two essential reasoning capabilities, knowledge about their own best position and awareness of the best position obtained by any particle in the neighbourhood. Individuals in the swarm exchange information about good positions and adjust their own positions as well as velocity according to these positions. PSO shares similarities with some evolutionary algorithms such as GA. However, unlike GA, PSO has no evolution operators, e.g., crossover and mutation.

2.5 Review of job shop scheduling

The job shop scheduling problem studied in this thesis is described as follows. Given a set of jobs and a set of machines, each job consists of a sequence of operations that has to be processed in a specific order. Each operation has to be performed on a given machine without interruption. The objective is to find the schedule that has the minimum makespan (the duration in which all jobs are completed), subject to the following constraints: each machine can handle at most one job at a time, and the operation precedence is respected for each job.

The job shop scheduling problem is difficult to solve optimally. This problem is not only NP-hard, it is also considered as one of the most computationally stubborn combinatorial problems.

In the past several decades researchers have employed many techniques to tackle the job shop scheduling problem. These solution techniques could be classified into two main groups, optimization and approximation methods (Jain and Meeran, 1999). Optimization approaches are among the earliest methods that have been used to solve the job shop scheduling problem. The Branch and Bound (Ashour and Hiremath, 1973) is an example of the optimization techniques.

In branch and bound a dynamically constructed tree representing all feasible schedules is searched. This method uses procedures and rules allowing large portions of the tree to be eliminated from the search until an optimal solution is found. For many

years, branch and bound was the dominant technique for solving job shop scheduling problem (Carlier and Pinson, 1989; Applegate and Cook,1991). Although this method is suitable for small problems, its application to medium and large size problems is limited as it requires significant computational time and enormous amount of memory.

Due to the limitations of the optimization techniques, a growing interest has been developing on the approximation methods to solve job shop scheduling problem. Shifting Bottleneck Procedure (Adams *et al.*, 1988; Mason *et al.*, 2002) is among the well-known approximation methods for the job shop scheduling problem. This procedure employs a sequence of decomposition, bottleneck identification, single-machine scheduling and re-optimization. Nonetheless, other research showed that this method suffers from difficulty in the re-optimization phase and infeasible solutions (Balas *et al.*, 1995).

The most popular approximation algorithms are Simulated Annealing (Van Laarhoven *et al.*, 1992), Tabu Search (Nowicki and Smutnicki, 1996; Dell'Amico and Trubian, 1993), and Genetic Algorithm (Mattfeld and Bierwirth, 2002; Yamada and Nakano, 1997; Della Croce *et al.*, 1995). These methods both in their basic forms and in hybrid with other methods have been widely applied to the job shop scheduling problem. Heuristic techniques proved to be efficient and could be applied to a wide range of combinatorial problems.

In recent years several other heuristics such as Greedy Randomized Adaptive Search (GRASP) (Binato *et al.*, 2001), Ant System (AS) (Colorni *et al.*, 1994), and Particle Swarm Optimization (PSO) (Xia and Wu, 2006) have been also applied to the job shop scheduling problem.

2.6 Review of flow shop scheduling

A permutation flow shop scheduling problem can be stated as follows. There are n jobs to be processed successively on m machines. Each job can be processed on no more than one machine at any time, while each machine can handle only one job and the processing of a job may not be interrupted. Moreover, it is assumed that the same job order is chosen on each machine. The objective is to find the schedule that has the minimum makespan (the duration in which all jobs are completed).

The problem is known to be NP-complete for three or more machines (Garey *et al.*, 1976). Therefore, the research has been concentrated mainly on developing heuristics to obtain near optimal solutions for the problem.

Generally, the solution techniques for the flow shop scheduling problem are classified as exact methods, constructive heuristics, and improvement approaches. The exact methods are among the earliest techniques that have been exercised to solve the flow shop scheduling problem. The main characteristic of the exact methods is their ability to obtain optimal solution for the problem under consideration. However, they are only suitable for small size problems because of their significant computational time requirements. Dynamic programming (Held and Karp, 1962), Johnson's rule and its extensions (Johnson, 1954), as well as the branch and bound method (Lageweg *et al.*, 1978; Carlier and Rebaï, 1996) are some examples of exact methods that have been applied to the flow shop scheduling problem.

According to Lourenço (1996), a constructive heuristic is "an algorithm that builds a sequence of jobs and once a decision is made, it is never changed". The constructive heuristics provide good solutions for even large size flow shop scheduling problems. However, they are unable to find optimal solution in most cases. Algorithms developed by Palmer (1965), Gupta (1971a and 1971b), Campbell *et al.* (1970), and Newaz *et al.* (1983) are typical constructive heuristics to solve the flow shop scheduling problem.

Finally, the third group of solution techniques for flow shop scheduling problems consists of improvement heuristics. This class of algorithms has been developed mainly to mitigate the local optimality problem associated with constructive heuristics. Many such approaches have been applied to flow shop scheduling problems. The most popular methods include simulated annealing (Osman and Potts, 1989; Obgu and Smith, 1990; Ishibuchi *et al.*, 1995; and Wang and Zheng, 2003), tabu search (Taillard, 1990; Nowicki and Smutnicki, 1996; Ben-Daya and Al-Fawzan, 1998; Grabowski and Wodecki, 2004), and genetic algorithms (Reeves, 1995; Chen *et al.*, 1995; Murata *et al.*, 1996; Reeves and Yamada, 1998). Ant Colony Optimization (also known as Ant Colony Systems) (Stützle, 1998), GRASP (Prabhaharan *et al.*, 2006), and Particle Swarm Optimization (Liao *et al.*, 2007) are some instances of the newly developed improvement algorithms that have been also applied to flow shop scheduling problems.

Among the improvement approaches, two types of methods, generic and specific neighbourhood approaches, could be distinguished (Framinan *et al.*, 2002). A specific neighbourhood approach uses problem-specific information to define the neighbourhood of a solution and may not be suitable for other problems. On the other hand, a generic neighbourhood approach is a mechanism that could be used in a variety of combinatorial optimization problems. Pairwise exchange is an example of a generic neighbourhood approach. Since the algorithms with specific neighbourhood approach incorporate problem-specific information to generate a neighbour, they usually perform better than those algorithms with general neighbourhood approach.

Given the above classification scheme, the simulated annealing algorithms of Osman and Potts (1989), Obgu and Smith (1990), Ishibuchi *et al.* (1995), the tabu search of Taillard (1990), and the genetic algorithms by Reeves (1995), Chen *et al.* (1995), and Murata *et al.* (1996) could be categorized as the methods based on generic neighbourhood. The tabu search of Nowicki and Smutnicki (1996) and that of Grabowski and Wodecki (2004) are considered as the techniques with specific neighbourhood.

2.7 Motivation and objectives

As mentioned above, only a limited number of studies have addressed the human related issues to improve manufacturing productivity. Among these issues, human boredom has often been overlooked. There are many reasons for this but the most important one is probably the subjective nature of these issues that make them difficult to quantify. Motivated by the existing gap in the literature and growing recognition of the importance of human factors at work, one of the goals of this research is to address the boredom issues in a manufacturing environment.

Furthermore, numerous studies have addressed the impact of machine utilization on manufacturing productivity and efficiency. More precisely, machine scheduling has been studied extensively in the literature in the past several decades. However, manufacturing productivity is still being negatively affected by the lack of efficient solution techniques for the machine scheduling problems. Problems in this domain are widely recognized as the most stubborn combinatorial optimization problems. The other goal of this study is to

design a new metaheuristic approach to solve combinatorial optimization and scheduling problems.

Therefore, the objectives of the thesis are to:

1) Develop a methodology and appropriate solution technique to model employee's boredom at work,
2) Develop a mathematical programming model for job rotation as a means to cope with employee's boredom at work,
3) Develop an efficient solution technique i.e., heuristic to solve the job rotation problem, and
4) Apply the solution technique to job scheduling problems.

Chapter 3

Modeling Job Boredom Using Bayesian Networks

Most researchers agree that boredom reflects an emotional behaviour (Farmer & Sundberg, 1986; Mikulas & Vodanovich, 1993; O'Hanlon, 1981). The evidence that boredom is a distinct emotional state has been provided by researchers who have developed empirical topologies of emotion (Smith and Ellsworth, 1985). Feeling bored at work is a common complaint; a large percentage of employees feel bored at least occasionally and some even feel bored much of the time. Individuals experiencing boredom usually have difficulty to keep their attention focused on work and may feel that time is passing very slowly.

Two self reporting scales have been commonly used to measure boredom at work. The first scale is provided by Grubb (1975) and includes 11 items arranged on a five-point scale. The second job boredom scale is suggested by Lee (1986). Lee's job boredom scale consists of 17 items also organized on a five-point scale. Lee's job boredom scale is presented in Appendix A. The reviewed literature in Chapter 2 indicates that thus far there is no instrument to quantitatively measure employee's boredom at work.

In view of the above, an attempt has been made to model job boredom using a probabilistic framework. The probabilistic framework is based on the state-of-the-art Bayesian Networks (BN). The proposed model could be used to measure and to predict employee's boredom at work. In this chapter, a static Bayesian Network is first introduced to capture the static characteristic of employee's boredom modeling. Then, based on the proposed static model a dynamic Bayesian network is presented to account for temporal aspect of boredom modeling.

3.1 Theoretical background

A Bayesian network (BN) is composed of a graph and a set of parameters. The graph consists of nodes and arcs which respectively represent variables and casual/influential relationships between the variables. The parameters in a Bayesian network specify the probability distribution associated with each variable presented in a set of *Conditional Probability Tables* (CPT). Figure 3-1 illustrates a simple example of BN. The example is adopted from Russell and Norvig (2003).

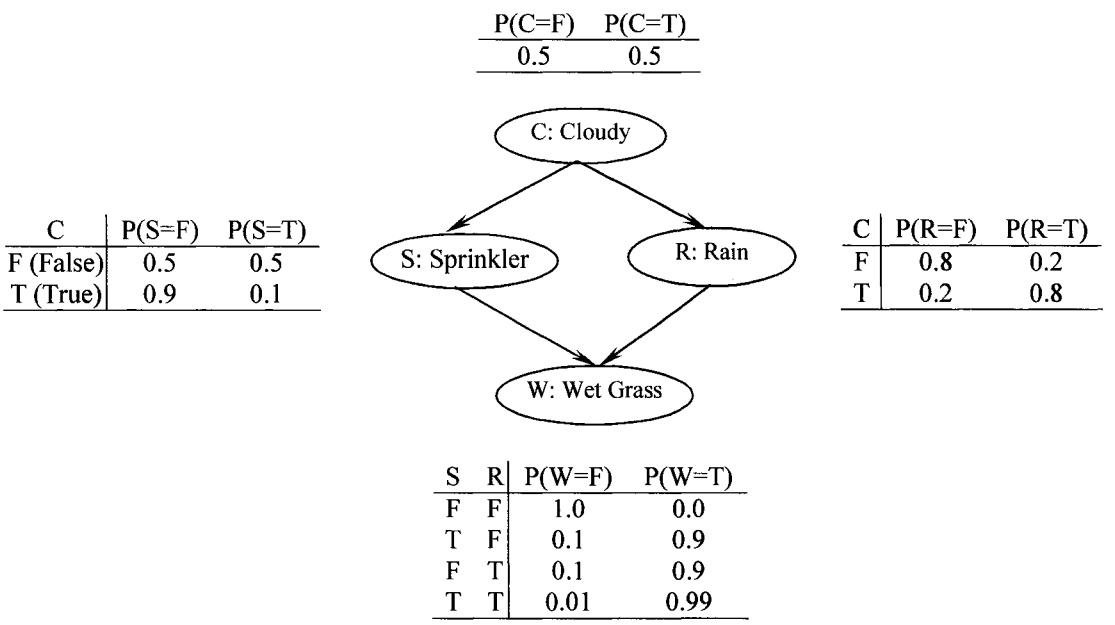


Figure 3-1. A simple Bayesian network (T: True and F: False)

In the example the event "grass is wet" (W=true) has two possible causes: either the water sprinkler is on (S=true) or it is raining (R=true). The strength of this relationship is shown in the table beside the node "W". For instance, the probability that grass is wet given the sprinkler is on and it is not raining is 0.9 (Pr (W=true | S=true, R=false) = 0.9) (second row). Since each row must sum to one hence, Pr (W=false | S=true, R=false) = 1 - 0.9 = 0.1. Furthermore, in the above BN example, an arc points from node "Cloudy" (C) to node "Rain" (R). C is called *parent* of R and R is called *child* of C. In addition to the graph structure shown in Figure 3-1, the local probability distribution associated with each node is presented as CPTs beside the nodes.

The main building block of the BN theory is Bayes' Theorem. This theorem is stated as follows:

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)}, \tag{3.1}$$

where $p(A|B)$ is the *posterior* probability of hypothesis A given data B, $p(B|A)$ is probability of data B given hypothesis A or the *likelihood* of the data, $p(A)$ is the *prior* probability of hypothesis A, and $p(B)$ is the probability of data B or *evidence*.

The two other important rules in BNs are the *expansion* rule or *marginal probability* and the *chain* rule. Assuming that A and B are random variables with k possible outcomes, the marginal probability is stated as

$$\begin{aligned} p(A) &= p(A|b_1)p(b_1) + p(A|b_2)p(b_2) + ... + p(A|b_k)p(b_k) \\ &= \sum_B p(A|B)p(B) \end{aligned} \tag{3.2}$$

The chain rule is derived by writing Bayes' Theorem in the following form, which is called the *product* rule:

$$\begin{aligned} p(A,B) &= P(A|B)p(B) \\ &= p(B|A)p(A) \end{aligned} \tag{3.3}$$

Successive application of the product rule yields the chain rule:

$$\begin{aligned} p(A_1,...,A_n) &= p(A_1,...,A_{n-1})p(A_n|A_1,..,A_{n-1}) \\ &= p(A_1,...,A_{n-2})p(A_{n-1}|A_1,..,A_{n-2})p(A_n|A_1,...,A_{n-1}) \\ &= p(A_1)p(A_2|A_1)...p(A_n|A_1,...,A_{n-1}) \\ &= p(A_1)\prod_{i=2}^{n} p(A_i|A_1,...,A_{i-1}) \end{aligned} \tag{3.4}$$

Using the chain rule, in a BN with n random variables (e.g., $A_1,\ldots, A_n$) the joint probability of the set of variables is given as

$$p(A_1,...,A_n) = \prod_{i=1}^{n} p(A_i|A_1,...,A_{i-1}) \tag{3.5}$$

The conditional independence relationships encoded in the Bayesian network state that a node A_i is conditionally independent of its ancestors given its parents π_i. Therefore,

$$p(A_1,...,A_n) = \prod_{i=1}^{n} p(A_i|\pi_i) \tag{3.6}$$

where $p(A_i|\pi_i)$ is called the *Local Probability Distribution* (LPD). The process of breaking up the joint probability distribution into local probability distributions is called *factorization*, which results in an efficient representation that supports fast inference. This is a property of BNs that plays a major contribution to its success.

3.2 Static Bayesian Networks for boredom modeling

3.2.1 Structure of the networks

Bayesian networks are capable of incorporating prior information (knowledge), explicitly modeling uncertainties, and modeling temporal aspect of a problem. Constructing a BN generally involves three main steps. The first step is to define the problem domain which includes identification of the relevant variables constituting the problem and their possible states. The second step is to determine the relationships among the variables. The third step is to specify the conditional probability values for each variable in the model.

The initial task in modeling employee's boredom as mentioned earlier is to identify all related variables. In this study the associated variables to job boredom are determined through conducting an intensive literature review over the subject. These variables could be divided into two distinguished groups of *causes* and *effects*.

Variables that may contribute to make someone feel bored at work include but are not limited to: organizational control practice, presence of other people, quality and quantity of the work, personality, capacity, and mental health. The observable data or the variables representing the consequences of boredom are identified as absenteeism, job dissatisfaction, accidents, and performance variation.

For boredom modeling, boredom is the target hypothesis variable that is intended to be inferred. The other contextual variables that could cause boredom, and those representing boredom effects i.e., symptoms of boredom, are named information variables. Definitions of the variables along with their possible states for boredom modeling are presented in Table 3-1.

Table 3-1. Variables of the static Bayesian networks

Variable		
Name	Definition	States
Work quantity	Work load status	Normal/under load
Work quality	type of work in terms of level of complication (simplicity), challenging, and skill requirements	Normal/Abnormal
Organizational control practice	the extend to which an organization constraint workers behaviour at work	Normal/Tight
Presence of other people	presence of co-workers in workplace	Yes/No
Work condition	characteristic of work in terms of both quantity and quality performed by individuals	Normal/Poor
Work environment	characteristic of workplace in which individuals perform their tasks	Fair/Poor
Personality	characteristic of a person as if required external stimulation to stay at good level of activation or he/she might be self-actuated	Extraversion/ introversion
Capacity	Ability to perform a task given worker's age, physical health, education, and experience	High/Low
Mental health	pathological cause of boredom	boredom-prone / nonboredom-prone
Person effect	effect of person on boredom given his/her personality, capacity, and mental health	High/Low
Boredom	a transient state in which individuals feel lack of interest in their current activity	High/Low
Absenteeism	absence from workplace	High/Low
Job satisfaction	worker's feelings or state-of-mind regarding the nature of their work	High/Low
Accidents	An unexpected and undesirable event, especially one resulting in harm (damage) to people (equipments) at workplace	High/Low
Performance variation	frequency of changes in worker's performance	High/Low
Age	-	Young/Old
Physical health	worker's physical health condition	Fit/Poor
Education level	State of worker's education level	High/Low
Experience	Worker's level of experience	High/Low

Upon identifying the variables, the relationship between variables is determined followed by assigning appropriate degree of influence between parent and child nodes. The degree of influence between two variables could be Strong (S), Medium (M), or

Weak (W). It is worthwhile to mention that the assigned degrees of influences are based on various conducted surveys that have measured the correlation between each variable and the target hypothesis, boredom. For instance, MacDonald and MacIntyre (1997) used an employee sample from several different job categories and found a significant, negative correlation between scores on the job satisfaction scale and boredom. Lee (1986) using a sample of clerical employees (N = 322, 95% female) found that high scores on the job boredom scale were significantly associated with lower job satisfaction.

Once the relationship between the variables is established, the graphical representation of BN for boredom could be constructed. The structure of the boredom model is illustrated in Figure 3-2. In this figure, the sign (+) or (-) represents the positive or negative relationship between the variables. For example, the work quality variable in Table 3-1 has two possible states (Normal and Abnormal). This variable has a strong positive relation with work condition. For example, an abnormal work quality will result a poor work condition (Figure 3-2).

The graph topology presented in Figure 3-2 is limited to the definitions of the important variables causing boredom and those identified as its effects.

3.2.2 Constructing the conditional probability tables

The next step in developing a BN for boredom is to specify the conditional probabilities associated with each node in the network. If data were available, special algorithms could be used to learn prior and conditional probabilities of the boredom model. However, this is not possible as such data in this case is not available. To alleviate such a problem the Qualitative Bayesian Networks (QBNs) (Wellman 1990, Druzdzel and Henrion 1993) has been proposed in the literature. In QBNs, special types of algorithms such as Qualitative Belief Propagation (Druzdzel and Henrion 1993) are implemented to propagate evidence through the networks using the degrees of influences. Nonetheless this approach still does not map degrees of influence to actual CPTs.

In this thesis, the following two techniques are utilized to parameterize the BN model for boredom. The first method is a combination of qualitative explanation approach and a weighting scheme (Daniel *et al.*, 2006). The main advantage of this

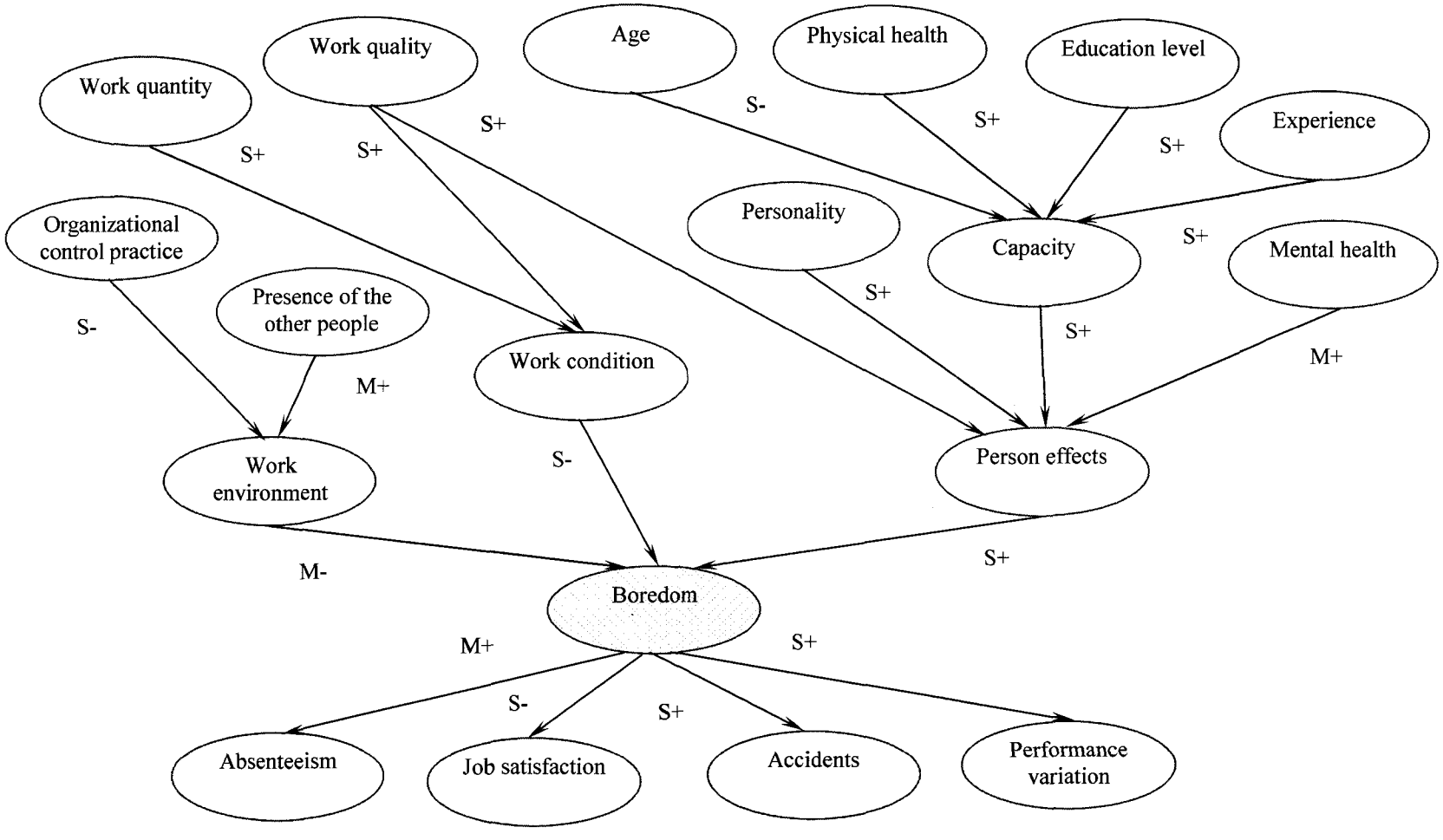


Figure 3-2. A Bayesian model for boredom at work (degree of influence: Strong (S), Medium (M), and Weak (W))

method over the QBNs (Wellman, 1990; Druzdzel and Henrion, 1993) is its ability to map degrees of influence to actual CPTs. The second method is based on the so called "noisy-OR" principle (Henrion, 1989) through which some conditional probabilities are inferred.

3.2.2.1 Qualitative explanation approach

To determine the CPT for each node, a threshold value that represents the maximum probability value that a child node could reach given the degree of influence of its parent node is assigned. For the strong, medium, and weak degrees of influence, the threshold values are decided to be 0.98, 0.8, and 0.6, respectively (Daniel *et al.,* 2006). Then, a weight value corresponding to each degree of influence is calculated based on the threshold value and the number of parents.

The weight values are determined according to the following formula

$$Weight\,Value = \frac{Threshhold\,Value - Base\,Value}{Number\,of\,Parents} \tag{3.7}$$

where

$$Base\,Value = \frac{1}{Number\,of\,Parents} \tag{3.8}$$

To illustrate, consider the work condition node in Figure 3-2. This node has two parents "work quality" and "work quantity". Given the degree of influence between the two parent nodes (i.e., work quality and work quantity) on the child node "work condition" (which is S+ for both parents), the weight values associated with both degree of influences are calculated as

$$(0.98\text{-}0.50) / 2 = 0.24$$

where the base value is (1/number of parents = 1/2= 0.5). The possible weight values for a child with two parents are presented in Table 3-2.

Finally, to determine all elements of the CPT for work condition node, the base value is added to the associated weight values given the presented evidences.

Table 3-2.Weight values for a child node with two parents

Degree of influence	Threshold	Weight
Strong (S)	0.98	0.24
Medium (M)	0.80	0.15
Weak (W)	0.60	0.05

Using the described method, the conditional probability table for work condition node is calculated as follows:

P (work condition = poor | work quality = abnormal & work quantity = under load) = 0.5 + 0.24+ 0.24 =0.98

P (work condition = Normal | work quality = abnormal & work quantity = under load) = 1-0.98 = 0.02

P (work condition = poor | work quality = Normal & work quantity = under load) = 0.5 + 0.24 = 0.74

P (work condition = Normal | work quality = Normal & work quantity = under load) = 1-0.74 = 0.26

P (work condition = poor | work quality = abnormal & work quantity = Normal) = 0.5 + 0.24 = 0.74

P (work condition = Normal | work quality = abnormal & work quantity = Normal) = 1-0.74 = 0.26

P (work condition = poor | work quality = Normal & work quantity = Normal) = 0.5

P (work condition = Normal | work quality = Normal & work quantity = Normal) = 1- 0.5 = 0.5

In the above illustration, a special case arises when there is evidence that both work quantity and work quality are normal. In such cases, one expects to obtain high conditional probability value of work condition = normal when both work quantity and work quality are normal and, by the same token, a low conditional probability of work condition = poor if the same evidence is provided. To alleviate this shortcoming, Daniel et al. (2006) suggested using the threshold values directly as the conditional probability values. In other words, the probability that work condition is normal given that both work

quantity and work quality are normal is assumed to be 0.98 and the probability that work condition is poor when both work quantity and work quality are normal will be 0.02.

The above method is used to determine the associated conditional probabilities of each node in the boredom model (Figure 3-2). The CPTs for each node are presented in tables 3-4 to 3-8. The prior probabilities are estimated subjectively and presented in Table 3-3.

Table 3-3.Prior probabilities

Node	State	Probability
Organizational control practice	Tight	0.05
	Normal	0.95
Presence of other people	Yes	0.95
	No	0.05
Work quality	Abnormal	0.05
	Normal	0.95
Work quantity	Under load	0.05
	Normal	0.95
Age	Young	0.40
	Old	0.60
Physical health	Fit	0.60
	poor	0.40
Education level	High	0.10
	Match	0.90
Experience	High	0.15
	Low	0.85
Personality	Extraversion	0.20
	Introversion	0.80
Mental health	Boredom-Prone	0.05
	Nonboredom-Prone	0.95

Table 3-4.Conditional probability for work condition node

Parent nodes		Work condition	
Work quality	Work quantity	Poor	Normal
Abnormal	Under load	0.98	0.02
	Normal	0.74	0.26
Normal	Under load	0.74	0.26
	Normal	0.50	0.50

Table 3-5.Conditional probability for work environment node

Parent nodes		Work environment	
Organizational control practice	Presence of other people	Poor	Fair
Tight	No	0.89	0.11
	Yes	0.74	0.26
Normal	No	0.65	0.35
	Yes	0.50	0.50

Table 3-6.Conditional probability for capacity node

Parent nodes				Capacity	
Age	Physical health	Education	Experience	High	Low
Low	Fit	High	High	0.9800	0.0200
			Low	0.7975	0.2025
		Low	High	0.7975	0.2025
			Low	0.6150	0.3850
	Poor	High	High	0.7975	0.2025
			Low	0.6150	0.3850
		Low	High	0.6150	0.3850
			Low	0.4325	0.5675
High	Fit	High	High	0.7975	0.2025
			Low	0.6150	0.3850
		Low	High	0.6150	0.3850
			Low	0.4325	0.5675
	Poor	High	High	0.6150	0.3850
			Low	0.4325	0.5675
		Low	High	0.4325	0.5675
			Low	0.5000	0.5000

Table 3-7. Conditional probability for person effect node

Parent nodes				Person Effect	
personality	Capacity	Mental health	Work quality	High	Low
Extraversion	High	Boredom-Prone	Underload	0.9350	0.0650
			Normal	0.7525	0.2475
		Nonboredom-Prone	Underload	0.7975	0.2025
			Normal	0.6150	0.3850
	Low	Boredom-Prone	Underload	0.7525	0.2475
			Normal	0.5700	0.4300
		Nonboredom-Prone	Underload	0.6150	0.3850
			Normal	0.4325	0.5675
Introversion	High	Boredom-Prone	Underload	0.7525	0.2475
			Normal	0.5700	0.4300
		Nonboredom-Prone	Underload	0.6150	0.3850
			Normal	0.4325	0.5675
	Low	Boredom-Prone	Underload	0.5700	0.4300
			Normal	0.3875	0.6125
		Nonboredom-Prone	Underload	0.4325	0.5675
			Normal	0.2500	0.7500

Table 3-8.Conditional probability for boredom node

Parent nodes			Boredom	
Work environment	Work condition	Person effect	High	Low
Poor	Poor	High	0.9201	0.0799
		Low	0.7034	0.2966
	Normal	High	0.7034	0.2966
		Low	0.4867	0.5133
Fair	Poor	High	0.7634	0.2366
		Low	0.5467	0.4533
	Normal	High	0.5467	0.4533
		Low	0.3300	0.6700

3.2.2.2 Leaky noisy-OR gate

The word *noisy* in the name, noisy-OR (Henrion, 1989), indicates that the causal interaction is not deterministic, meaning that any cause can produce the effect with some probability. The second part of the name – *OR* determines the manner in which independent causes are combined to produce their common effect (Zagorecki and Druzdzel, 2004). The underlying assumptions for applying a (binary) noisy-OR gate could be stated as follow. Given the binary variables $A_1, A_2, A_3, \ldots, A_n$ representing all the possible causes of binary variable B, each of the causes A_i has a probability p_i of being sufficient to produce effect B in the absence of all other causes, and the ability of each cause A_i to produce that effect is independent of the presence of other causes (see Figure 3-3). These assumptions aim to determine the entire conditional probability distribution with only n parameters $p_1, p_2, \ldots, p_n$ where p_i represents the probability that effect B will be true if cause A_i is present and all other causes A_j, $j \neq i$, are absent. In other words,

$$p_i = P\left(b|\bar{a}_1, \bar{a}_2, \bar{a}_3, \ldots, a_i, \ldots, \bar{a}_{n-1}, \bar{a}_n\right)$$

where $\bar{a}_i$ and a_i represent the two outcomes of the binary variable A_i. The probability of b i.e., an outcome of binary variable B, given a subset $\mathbf{A}_s$ of the A_is which are present is given by the following formula (Henrion, 1989)

$$P\left(b|\mathbf{A}_s\right) = 1 - \prod_{i:a_i \in \mathbf{A}_s} \left(1 - p_i\right) \tag{3-9}$$

The complete CPT of B conditional on its predecessors $A_1, A_2, A_3, \ldots, A_n$ could be derived using the above formula.

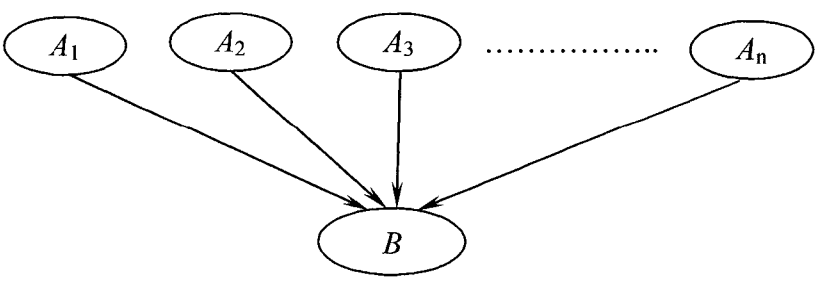


Figure 3-3. Graphical representation of the Noisy-OR gate

Henrion (1989) proposed an extension of the binary noisy-OR gate for situations where the effect variable can be true even if all its causes are false. The extended model is called *leaky* noisy-OR gate. Leaky noisy-OR is applicable to the situations where a model does not capture all possible causes of B. This can be modeled by introducing an additional parameter p_l, called the *leak probability*, the combined effect of all unmodeled causes of B,

$$p_l = P\left(b|\bar{a}_1, \bar{a}_2, \bar{a}_3, ..., \bar{a}_{n-1}, \bar{a}_n\right)$$

p_l represents the probability that effect B will occur in absence of any of the causes that are modeled explicitly.

In the leaky noisy-OR gate $p_i\,(i\neq 0)$ no longer represents the probability that A_i causes B given that all the other causes are absent, but rather the probability that B is present when A_i is present and all other explicit causes (all the A_j's such that $j\neq i$) are absent. Let p'_i be the probability that B will be true if A_i is present and every other cause of B including unmodeled causes are absent, then (Díez and Druzdzel, 2006)

$$1 - p'_i = \frac{1 - p_i}{1 - p_l} \tag{3-10}$$

From the above equation, we have

$$p_i = p'_i - (1 - p'_i)p_l \tag{3-11}$$

It follows that the probability of B given a subset A_s of a_i which are present is given in the leaky noisy-OR gate by the following formula:

$$P(B|A_s) = 1 - (1 - p_l) \prod_{i:a_i \in \mathbf{A}_s} \frac{1 - p_i}{1 - p_l} \tag{3-12}$$

Diez (1993) proposed an alternative way of eliciting the parameters of a leaky noisy-OR gate, which calls essentially to asking the expert for the parameters p'_i as defined by Equation (3-10). The difference between the two proposals has to do with the leak variable. While Henrion's parameters p_i assume that the expert's answer includes a combined influence of the cause in question and the leak, Diez's parameters p'_i explicitly refer to the relationship between the cause in question and the effect with the leak absent.

If the noisy-OR parameters are learned from data, Henrion's definition is more convenient, as the observed frequencies include the leak, which is always present by definition (Zagorecki and Druzdzel, 2004).

The noisy-OR gates are often utilized to describe the interaction between several causes and their common effect. As stated earlier, the main assumptions in using noisy-OR gate to approximate CPTs is that causes A_i are each sufficient to cause B in absences of other causes and all causes (parents) are independent of each other in terms of their abilities to influence the effect variable (child). Examining the boredom model with regard to the above assumptions, it turns out that all of the CPTs except for the *Capacity* node could be approximated by the noisy-OR model. The CPTs for the capacity node are determined using the method presented in section 3.2.2.1.

To determine the CPTs, the net probability values of each noisy node (given cause A_i is present and other explicit causes are absent) have to be first determined. The degree of influence between the two variables is used to approximate these probability values. For the strong, medium, and weak degrees of influence, the probability values are assumed (arbitrary) to be 0.80, 0.6, and 0.4, respectively. The net probability values as well as the leak values for noisy nodes are presented in Table 3-9 to Table 3-12.

Table 3-9. Net probability values for the Work environment node

	Parent		Leak
	Organizational control practice	Presence of other people	
State	Tight	No	
Poor	0.8	0.6	0.1
Fair	0.2	0.4	0.9

Table 3-10. Net probability values for the work condition node

	Parent		Leak
	Work quality	Work quantity	
State	Abnormal	Under load	
Poor	0.8	0.8	0.1
Normal	0.2	0.2	0.9

Table 3-11. Net probability values for the person effect node

	Parent			Leak
	Personality	Capacity	Mental health	
State	Extraversion	High	Boredom-prone	
High	0.8	0.8	0.6	0.1
Low	0.2	0.2	0.4	0.9

Table 3-12. Net probability values for the boredom node

	Parent			
	Work environment	Work condition	Person effect	Leak
State	Poor	Poor	High	
High	0.6	0.8	0.8	0.1
Low	0.4	0.2	0.2	0.9

3.2.3 Implementation and inference

The above BN models for boredom are implemented using the GeNIe environment developed by Decision Systems Laboratory, University of Pittsburgh (http://genie.sis.pitt.edu/). A snapshot of each model without presenting any evidence is presented in Figure 3-4 and 3-6. According to the information presented in these figures, the prior probability of boredom is 61% for the model that uses qualitative explanation approach and 57% for that with noisy-OR gate parameterization scheme. These values represent the average boredom level for an employee working in a relatively fair environment and with a normal working condition.

Inference or model evaluation is the procedure by which the probabilities of outcomes in a BN model are updated based on the existing relationship between variables and the available evidence about the situation. The information about recent event or observation is added to the model by fixing or "instantiating" the state of a variable(s) according to the observation(s). In the absence of real data, an alternative way to evaluate a BN model is to develop a number of scenarios or cases describing a set of phenomena that may occur in real world.

To evaluate the developed BN models, several scenarios have been developed and tested. In one scenario, for instance, it is assumed that in a particular situation there is evidence indicating that organizational control practice is *Tight*, the type of employee's personality is *Extraversion*, and the work quality is *Abnormal*. Figures 3-5 and 3-7 illustrate such BN models after the evidence is propagated through the models. The expected level of boredom measured by the model with qualitative explanation approach for the described employee is 77% (Figure 3-7). While the expected level of boredom for the same employee measured by the model with noisy-OR approach is 96% (Figure 3-7).

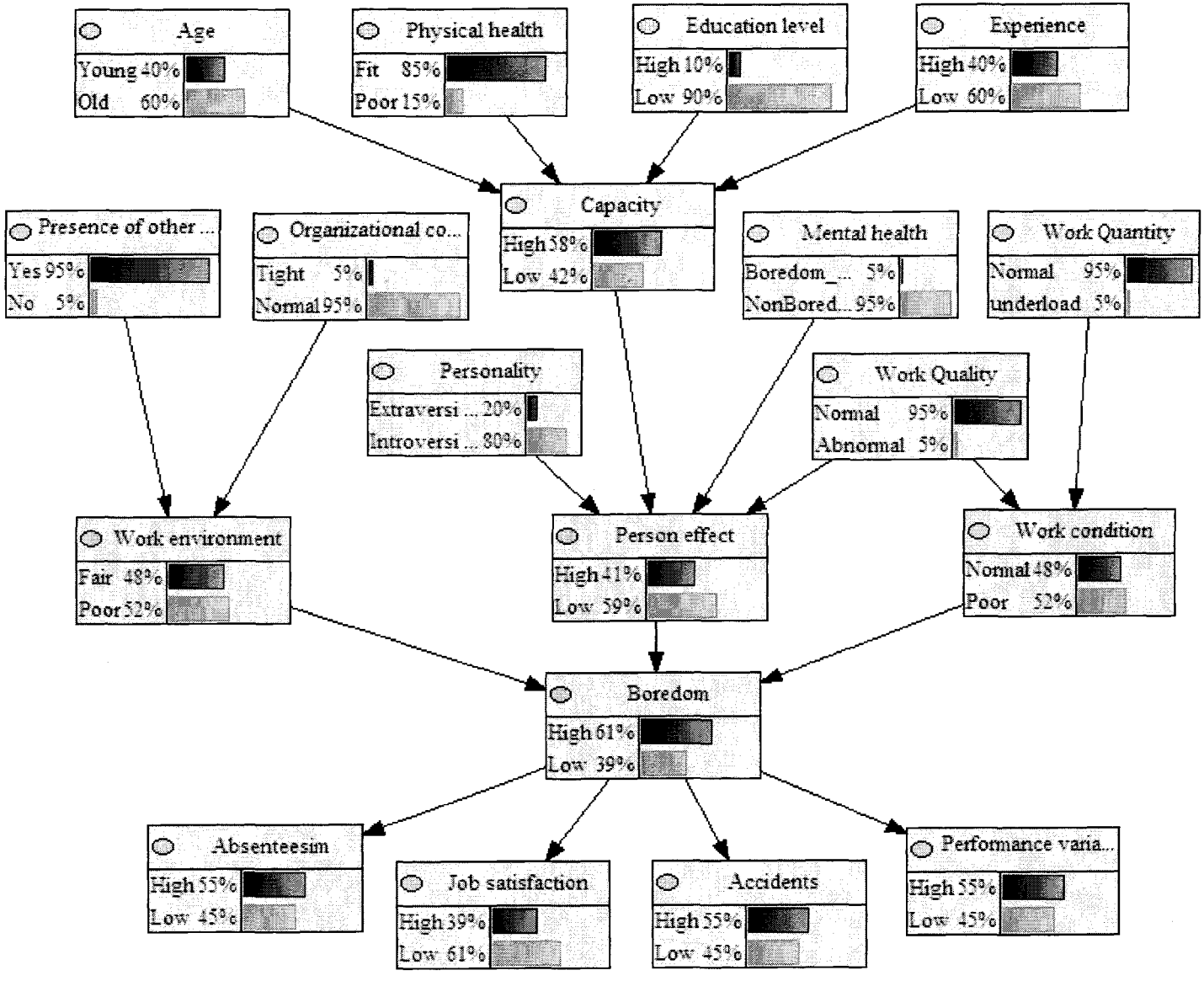


Figure 3-4. A snapshot of the updated boredom model before presenting evidence (qualitative explanation approach)

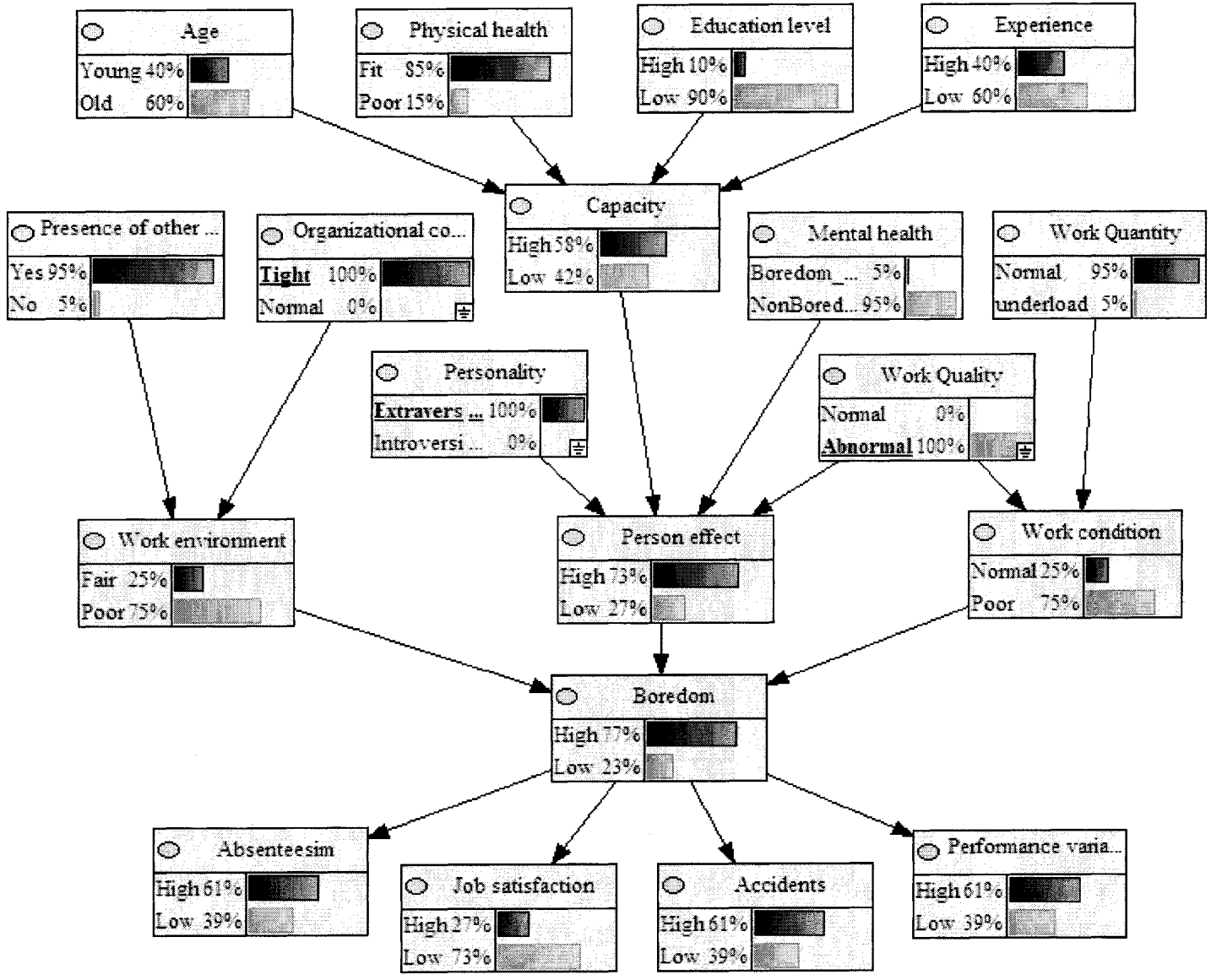


Figure 3-5. A snapshot of the updated boredom model after presenting the evidence (qualitative explanation approach)

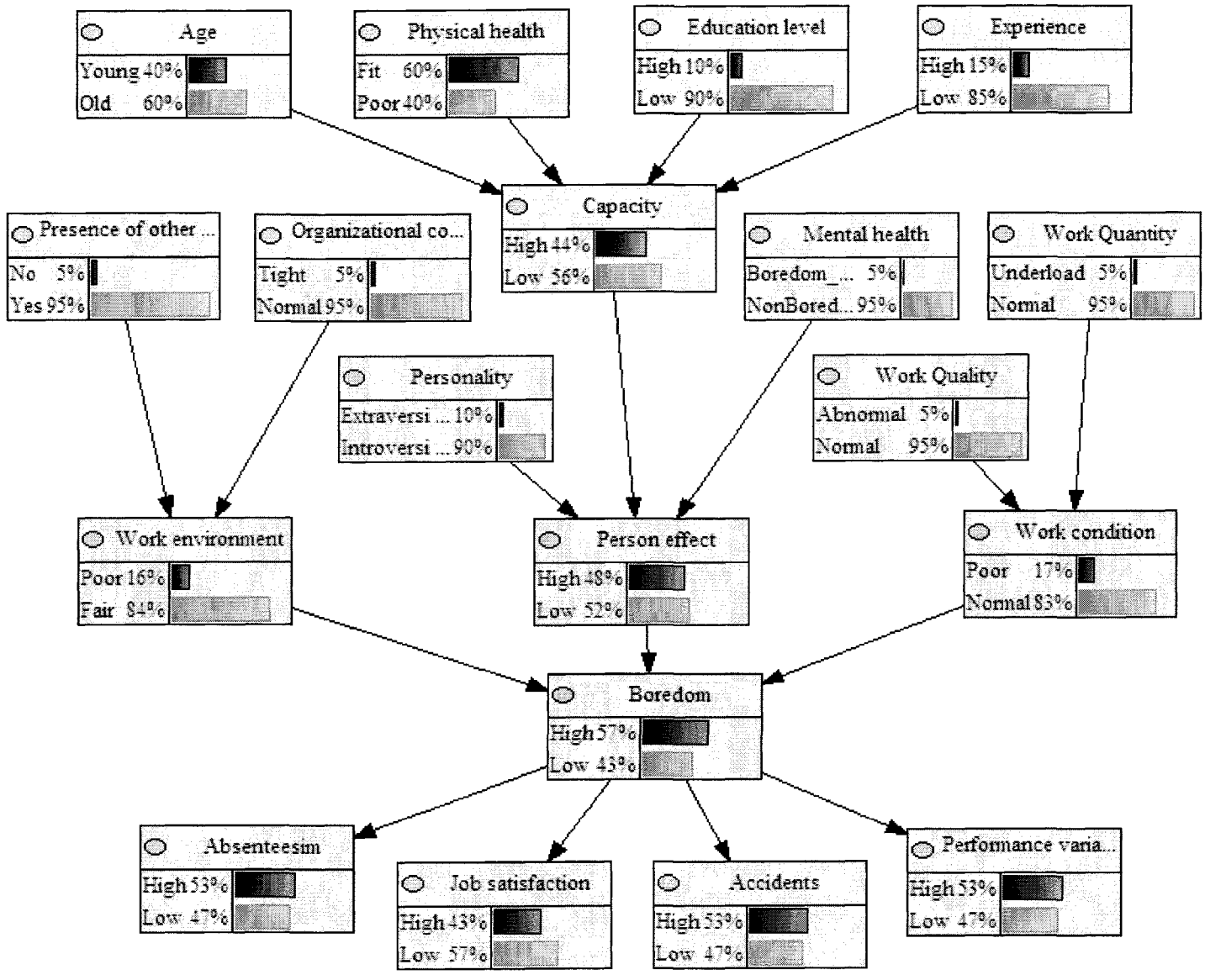


Figure 3-6. A snapshot of the updated boredom model before presenting evidence (noisy-OR gate)

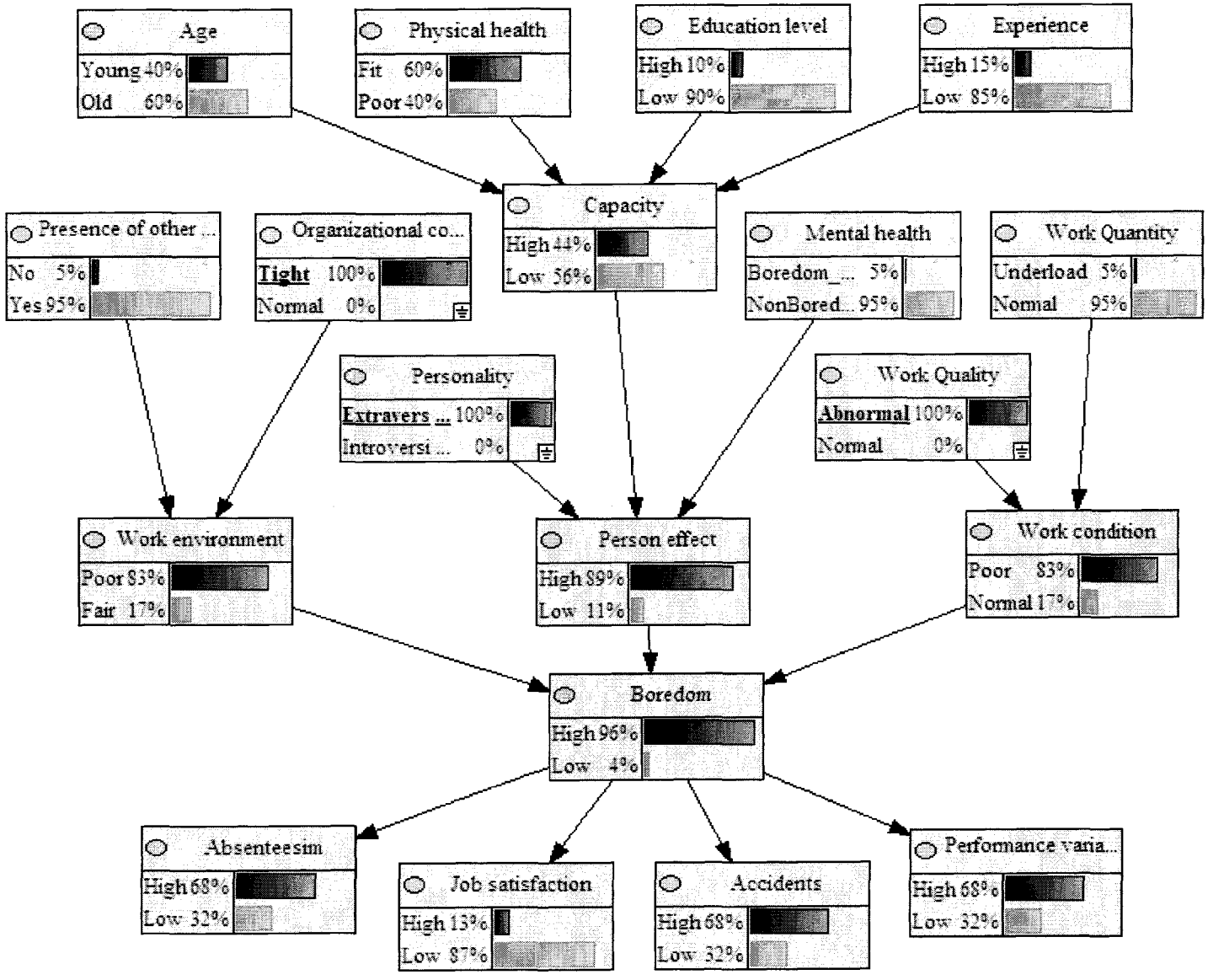


Figure 3-7. A snapshot of the updated boredom model after presenting the evidence (noisy-OR gate)

Theoretically, in a Bayesian model with a total number of 14 instantiable variables each having two states, there will be 2^{14} possible inference results. The results of 17 cases related to boredom node are summarized in Table 3-13. Table 3-13 also compares the inference results captured by the two proposed Bayesian models. The models differ in how their CPTs are specified.

Table 3-13 .Inference results of the boredom

Evidences	Boredom	
	Model with qualitative explanation approach	Model with Leaky noisy-OR gate
No evidence	0.61	0.57
Education level (high)	0.62	0.64
Experience (High)	0.62	0.64
Work quantity (under load)	0.66	0.83
Mental health (Boredom prone)	0.64	0.74
Personality (extroversion)	0.64	0.80
Organizational control practice (Tight)	0.65	0.76
Work quality (Abnormal)	0.70	0.83
Absenteeism (High)	0.79	0.75
Performance variation (High)	0.79	0.75
Accidents (Low)	0.40	0.36
Absenteeism (High) and Work quality (Abnormal)	0.84	0.92
Experience (High), organizational control practice (Tight), personality extroversion, and work quantity (under load)	0.76	0.96
Work quality (Abnormal), experience (High), and performance variation (High)	0.85	0.93
Absenteeism (High), Accidents (High), and performance variation (High)	0.95	0.94
Job satisfaction (High) and Accidents (Low)	0.07	0.06
Performance variation (Low)	0.40	0.36

3.3 Boredom modeling with dynamic Bayesian Networks

Boredom in previous time instant could affect boredom in the present time. Furthermore, it is more realistic to consider the persistent presence of boredom effects such as

absenteeism or accident over time (instead of the presence of the data in a particular time instant) to detect boredom. Therefore, it is important to consider the temporal (dynamic) aspect of the boredom.

A Dynamic Bayesian Networks (DBN) allows monitoring and predicting boredom by integrating boredom evidences both spatially and temporally. Figure 3-8 illustrates the structure of a simple DBN. In this figure, the hypothesis, hidden state, and observation nodes are represented by A^t, S^t, and B^t respectively. Furthermore, the horizontal arrows connecting nodes show the temporal dependencies, and the vertical arrows indicate the casual dependencies.

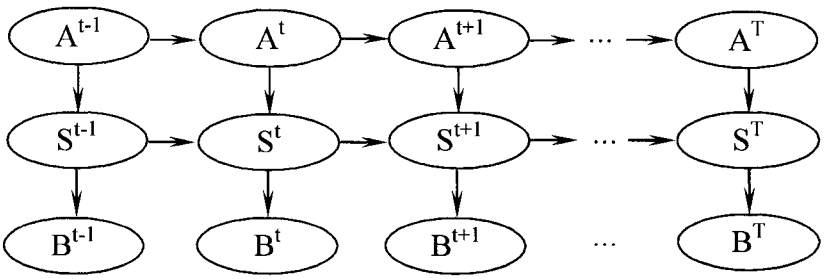


Figure 3-8. Generic structure of dynamic Bayesian Networks

Most of the studies on DBNs share two common assumptions. The first one is Markov assumption, which assumes that the future is conditionally independent of the past given the current state. The other is time-invariant assumption, i.e., the transition probabilities are independent of time and do not change with time.

In general, there are three approaches to develop a dynamic Bayesian Networks (Mihaljlovic and Petkovi, 2001): (1) models that use static BNs and formal grammars to represent temporal dimension which is also known as probabilistic temporal networks (PTNs); (2) models that use a mixture of probabilistic and nonprobabilistic frameworks; and (3) models that introduce temporal nodes into static BNs structure to represent time dependence. The third approach has been widely used to develop DBNs in the literature.

The proposed dynamic Bayesian model for boredom is based on the third approach. In this approach, a time slice is used to represent a snapshot of an evolving temporal process at a time instant. Then the DBN's is considered as a sequence of time slices each

representing the state of the variables at a particular point of time. These time slices are interconnected by temporal relations, which are represented by the arcs joining particular variables from two consecutive slices.

According to the DBN topology presented in Figure 3-8, to specify a complete dynamic Bayesian network, a number of probabilities including $p(A^t|A^{t-1})$, $p(S^t|S^{t-1})$, $p(S^t|A^t)$, and $p(B^t|S^t)$ have to be determined. All of these probabilities with the exception of transition probabilities characterize the static aspect of DBNs. Therefore, they remain the same for all time slices as the SBNs. As a result, the same parameters of the static BN model could be used for specifying the DBN probability distributions.

Based on the above considerations, the DBN model for boredom is constructed and illustrated in Figure 3-9. In this model the transitional probabilities are specified subjectively and presented in Table 3-14. The remaining probabilities are adopted from previously described SBN.

In order to test the initial DBN model, the following scenario is generated. Assume that based on previously recorded data, it is expected that the level of absenteeism for a worker fluctuates between high and low (level) every two time slices in the next 10 time slices. Figure 3-10 shows a snapshot of the model after the evidence has been propagated through the model. Concentrating on the boredom node, Figure 3-11 compares the level of the employee's boredom before and after the evidence has been propagated through the model. According to the information presented in this figure, if no evidence is presented to the model, the level of boredom for the described employee will stay in a steady level of about 60% for the entire nine time slices. However, after propagating the evidence through the model, the level of boredom starts fluctuating between a high level of 80% and a low level of 45% every two time slice.

Table 3-14.Transitional probabilities for boredom node

Parent nodes				Boredom	
Work environment	Work condition	Person effect	Boredom(t-1)	High	Low
Poor	Poor	High	High	0.9201	0.0799
			Low	0.9200	0.0800
		Low	High	0.7034	0.2966
			Low	0.7000	0.3000
	Normal	High	High	0.7034	0.2966
			Low	0.7000	0.3000
		Low	High	0.4867	0.5133
			Low	0.5000	0.5000
Fair	Poor	High	High	0.7634	0.2366
			Low	0.7500	0.2500
		Low	High	0.5467	0.4533
			Low	0.5500	0.4500
	Normal	High	High	0.5467	0.4533
			Low	0.5500	0.4500
		Low	High	0.3300	0.6700
			Low	0.7000	0.3000

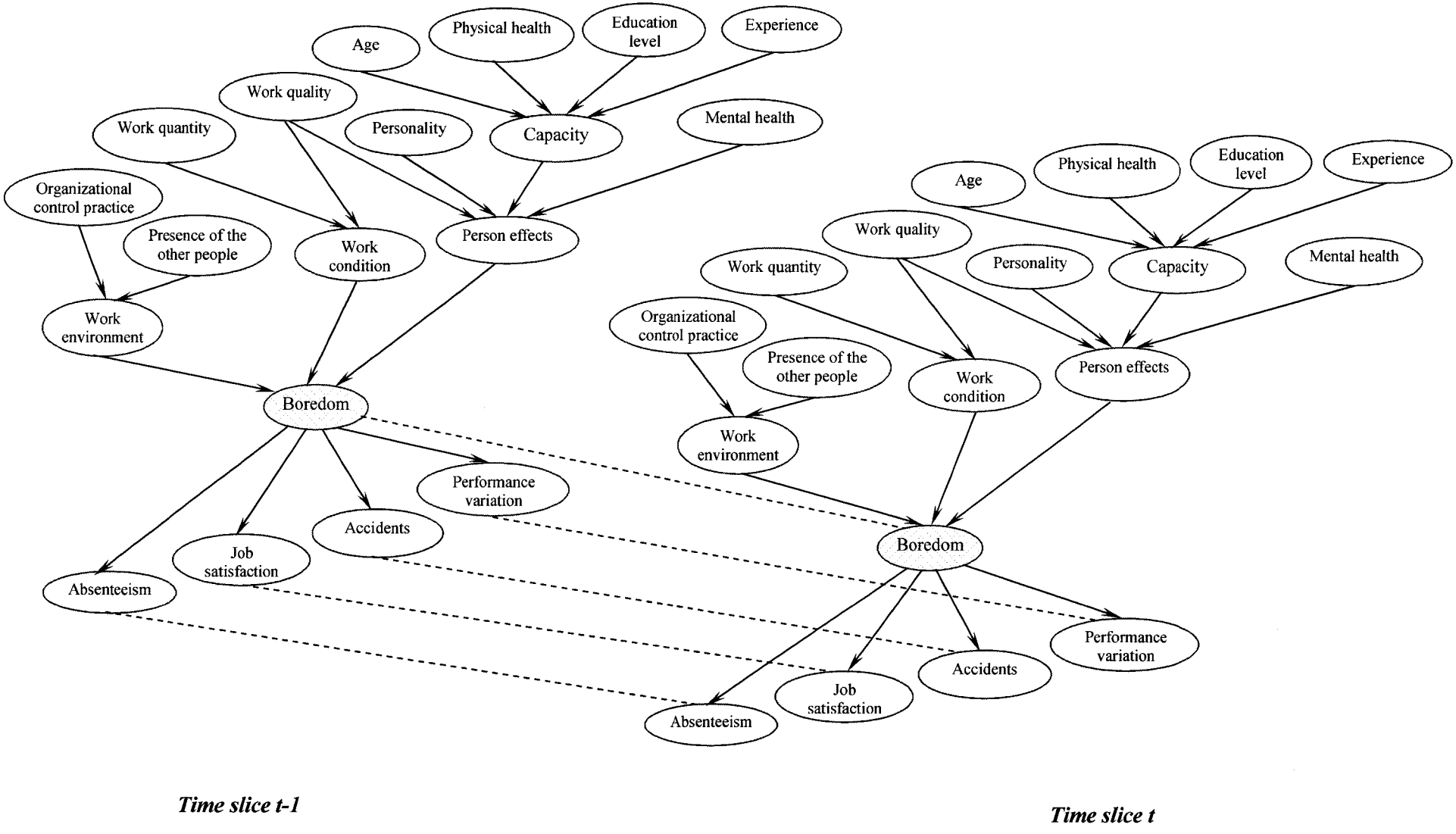


Figure 3-9. A dynamic Bayesian model for boredom at work

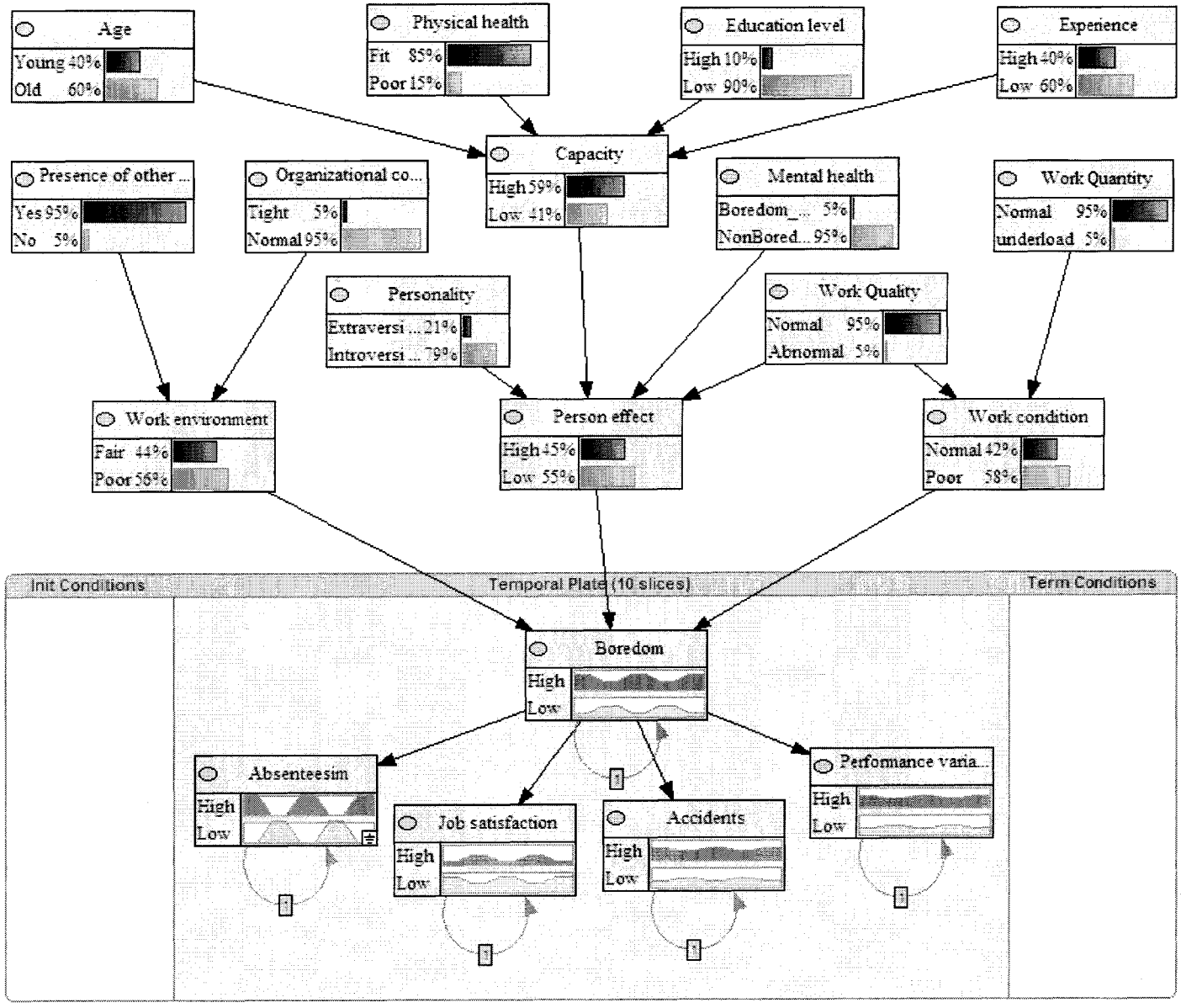
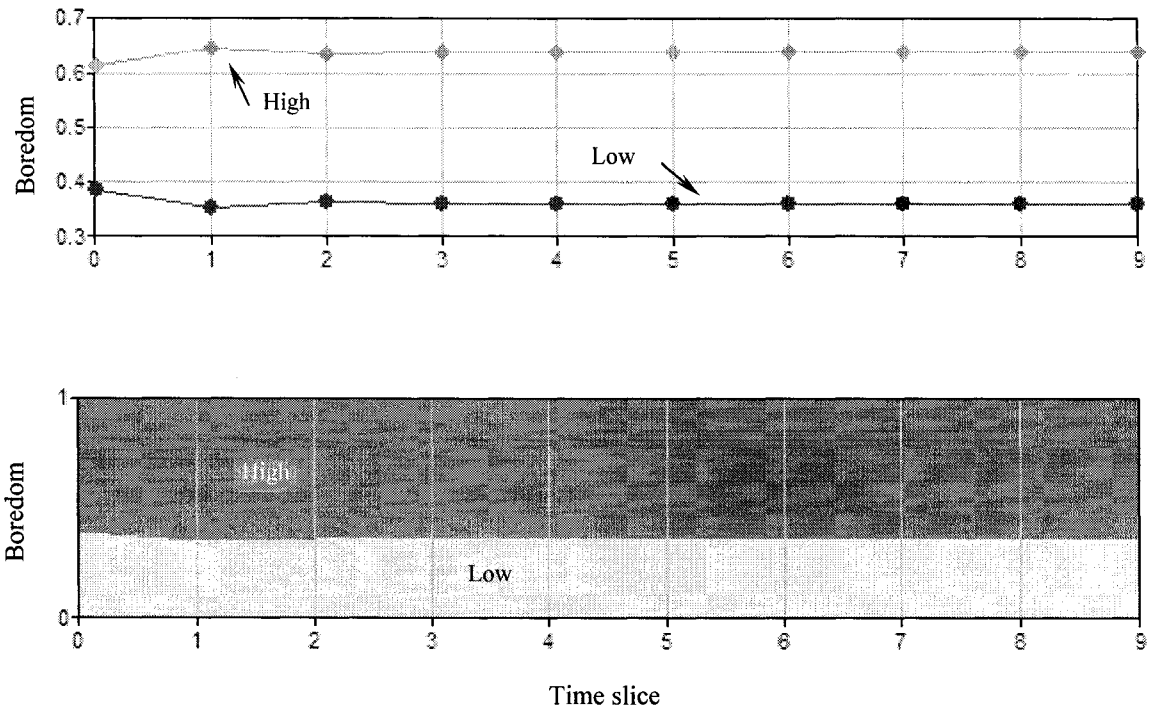
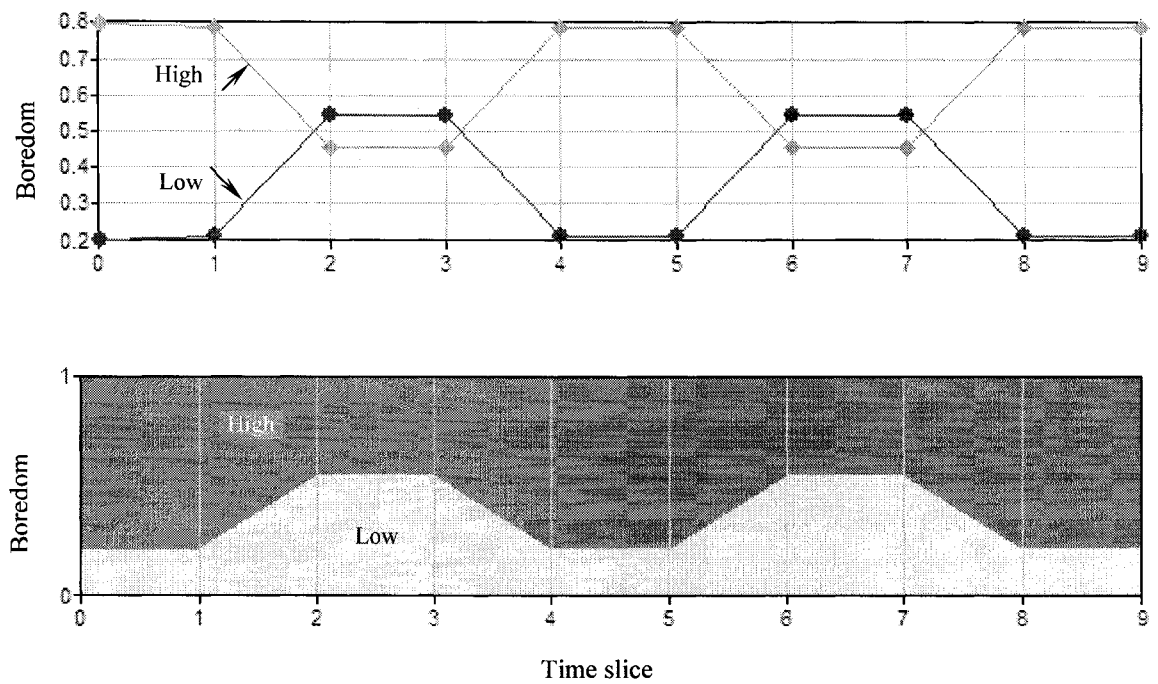


Figure 3-10. A snapshot of the dynamic Bayesian model after presenting the evidence



(a) Predicted level of boredom *before* the evidence is propagated



(b) Predicted level of boredom *after* the evidence is propagated

Figure 3-11. Comparison of boredom levels before and after evidence is propagated

Chapter 4

Modeling Job Rotation in Manufacturing Systems: The Impact of Workers Boredom and Skill Variations

Technical aspects of manufacturing systems have been the focus of enormous studies in the past several decades. However, the human related issues have not received adequate attention. Human aspects in a manufacturing firm such as boredom are still a neglected issue as pointed out by Fisher (1993) more than a decade ago. Job rotation could be an alternative strategy to cope with human boredom at work. The benefits of job rotation to employees and firms have been widely recognized in the literature. Those include reduced boredom, work related injuries and absenteeism as well as increased job satisfaction, However, the research on how to implement a job rotation plan has been very limited. Though some studies perceived job rotation as a means to reduce worker's physical work related traumas, they did not consider the effect of worker skill variations. Furthermore, despite the proven importance of emotions such as boredom at workplace, employee boredom has not been an explicit objective of any job rotation problem in accessible literature.

In this chapter, a methodology to formulate employee's boredom and skill variations are presented. A mathematical programming model is then introduced for job rotation in a manufacturing cell. To reduce the complexity of the model, a linear version of the proposed model is also presented and discussed.

4.1 Problem description

The problem considered herein investigates job rotation strategies in a manufacturing cell. In the manufacturing cell under consideration, a set of I workers are performing a set of J operations during a production planning horizon T^H. Each operation is performed by a single worker across each of the time units in the production planning horizon and each worker performs only one operation in each unit of time.

45

4.2 Assumptions

The model is subject to the following assumptions:

- Workers will receive a certain level of training prior to each assignment.
- The initial skill level of workers for all operations is the same.
- The maximum, minimum, upper bound, and lower bound level of skills are identical for all operations and all workers.
- Information pertaining to worker's learning, forgetting, boredom, and motivation slopes are available or can be estimated.
- products could be assumed identical.
- Each workstation is dedicated to a particular operation.

4.3 Methodology

4.3.1 Learning and forgetting curves

A learning curve could be defined as a graph that reflects the fact that as workers repeat their jobs, they improve performance. Industrial learning curve has been studied by many researchers in the past several decades. This phenomenon was reported by Wright (1936), who explained how the direct labour cost for producing airplanes decreased as the number of planes produced increase. The idea of the learning curve is that improvement occurs because workers learn how to do a job better as they produce more and more units. Nevertheless, it is generally accepted that other related factors such as job redesign, work and time analysis, and worker motivation also improve performance over time.

A power learning curve establishes an exponential relationship between the time required to produce the n^{th} unit and the cumulative production (Russell and Taylor, 1998). It could be presented as

$$y_n = y_1 n^b \tag{4.1}$$

where y_n is the time required to produce the n^{th} unit, y_1 is the time required to produce the first unit, n is the cumulative number of units produced, r is the learning curve coefficient, and b is the learning slope given by

$$b = \ln r / \ln 2 \tag{4.2}$$

During a learning process, if production is halted for a period of time, forgetting phenomenon may occur. In comparison to the learning curve, modeling of forgetting has not received much attention in literature (Kher, 1999; Shafer *et al.*, 2001). Several studies have acknowledged the existence of forgetting phenomenon in practical setting. Globerson *et al.* (1989) conducted a laboratory experiment to describe and analyze the forgetting phenomenon. The result of experiment indicated that forgetting of a task is a function of the break length and the level of experience gained prior to the interruption. Bailey (1989) studied the relearning phenomenon in a laboratory setting. Using a single break in learning process, the study showed that the degree of forgetting a task depends on the nature of the task being performed, forgetting rate, level of learning prior to the interruption, and the length of interruption interval. Jaber and Bonny (1996) developed a mathematical model for the relationship between learning and forgetting phenomena assuming that time for total forgetting is invariant of the experience gained prior to interruption. Jaber and Bonny (1996) showed that the forgetting slope is mathematically dependent on the learning slope, the quantity produced to date, and the minimum break at which total forgetting occurs. The authors also investigate the effect of forgetting phenomena on firm's productivity, labour cost, and total inventory system cost. Jaber and Kher (2004) later extended the work on learning forgetting curves by relaxing the assumption of a fixed time for total forgetting.

4.3.2 Skill improvement and deterioration curves

Generalizing the concept of learning and forgetting phenomena, the following relationship can be established between the amount of time a worker performs a particular operation and his (her) skill improvement. Once a worker is assigned to a workstation, his (her) skill improves as he (she) performs the same operation for a consecutive period of time (Figure 4-1). Worker's skill improvement may depend on the initial or remnant of the skill (if the worker is re-assigned to an operation), the length of the assignment, and the worker's slope of learning. Therefore, the corresponding skill improvement formula can be presented as:

$$S_{i,j,t} = S^{\max} - \left(S^{\max} - S^{Rem}_{i,j,a^k}\right) e^{\beta_i\left(t-a^k_{i,j}\right)} \tag{4.3}$$

where $S_{i,j,t}$ is the skill level of worker i performing operation j at time t, $S^{\max}$ is a theoretical maximum level of skill, β_i is the learning slope of worker i, $a^k_{i,j}$ is the time at which the k^{th} assignment of worker i has begun at workstation j (*i.e.*, time at which worker i transfers from workstation j-1 to j), and S^{Rem}_{i,j,a^k} is the remnants of worker's skill at $a^k_{i,j}$ which was gained by performing operation j in the past assignment (s). The underlying assumption in developing equation (4.3) is that, like many other natural phenomena, worker's skill improvement and deterioration follow an exponential pattern.

Using equation (4.2), the worker slope of learning could be expressed as:

$$\beta_i = \frac{\ln r_i}{\ln 2}, \tag{4.4}$$

where r_i is worker i learning coefficient. In general, r_i, the worker's learning curve coefficients could be measured using historical data. For the case of cell manufacturing, several methods have been proposed in the literature. For instance, Brown and Heathcote (2003) proposed a weighted average method to determine (part) family's learning rate. William *et al.* (2008) presented a new method for calculating the learning rate for a family of parts using a matrix-based approach.

According to equation (4.3), at infinite time (*i.e.*, $t \to \infty$) the skill of worker i performing operation j reaches the maximum level $S_{i,j,\infty} = S^{\max}$. However, achieving the maximum level of skill in infinite time is unrealistic. Therefore, a notion called

achievable skill level or *skill upper bound* is introduced. This notion is represented by S^{UB}. The relationship between $S^{\max}$ and S^{UB} can be expressed as

$$S^{UB} = S^{\max} - \varepsilon_1 \tag{4.5}$$

where ε_1 is skill upper bound *threshold* value. The time at which worker's skill reaches the upper bound level is represented by $t_{i,j}^{A}$ (Figure 4-1).

Furthermore, at time zero, i.e., $t = 0$, the skill of worker i is equal to $S_{i,j,0}^{Rem}$ that corresponds to the worker's initial skill level (i.e., $S_{i,j,0} = S_{i,j,0}^{Rem} = S^{Initial}$). It is worthwhile to mention that during a production horizon if a worker is assigned to a workstation for the first time, the remnant of worker's skill for that particular operation, i.e., S_{i,j,a^1}^{Rem}, is also substituted by $S^{Initial}$.

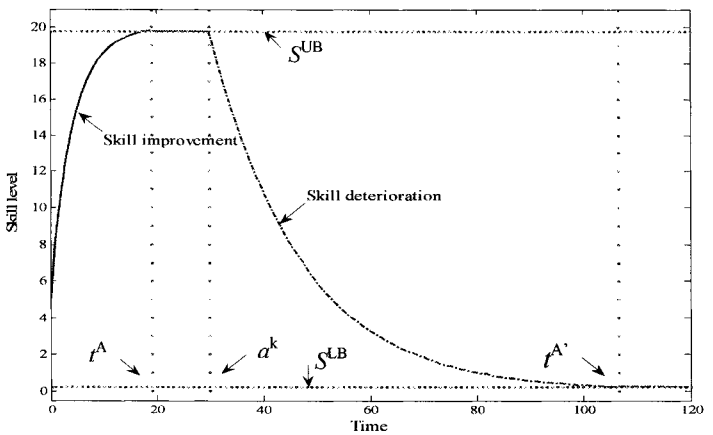


Figure 4-1. Skill improvement and deterioration curves

The remnant of skill, S_{i,j,a^k}^{Rem}, is the portion of the previously gained skill that has not been yet affected by forgetting phenomenon. The extent of worker's remnant of skill at any time may depend on the elapsed time between the current time, *t*, and the time at which the worker's experience with that specific operation was last interrupted, the worker's skill level at the time of interruption, and the worker's forgetting slope.

Hence, the corresponding skill deterioration formula could be presented as

$$S_{i,j,t}^{Rem} = S^{\min} + (S_{i,j,d^L} - S^{\min}) e^{\gamma_i (t - d_{i,j}^L)} \tag{4.6}$$

where $S^{\min}$ is the theoretical minimum level of skill, S_{i,j,d^L} is the skill level of worker *i* when departed (last time) from workstation *j*, γ_i is the forgetting slope associated with worker *i*, and $d_{i,j}^L$ is the time at which worker *i* has departed from workstation *j*.

Assuming that the minimum level of skill is zero, i.e., $S^{\min} = 0$, the skill deterioration formula could then be simplified as

$$S_{i,j,t}^{Rem} = S_{i,j,d^L} e^{\gamma_i (t - d_{i,j}^L)} \tag{4.7}$$

Similar to equation (4.4), the forgetting slope could be computed as

$$\gamma_i = \frac{\ln f_i}{\ln 2}, \tag{4.8}$$

where f_i is the forgetting coefficient of worker *i*.

According to equation (4.6), at infinite time ($t = \infty$) the remnant of skill *j* reaches the theoretical minimum level $S^{\min}$ (i.e., $S_{i,j,\infty}^{Rem} = S^{\min} = 0$). The *achievable lower level of skill* or *skill lower bound* is represented by S^{LB}. Therefore, the relationship between $S^{\min}$ and S^{LB} can be established as

$$S^{LB} = S^{\min} + \varepsilon_2 \tag{4.9}$$

where ε_2 is the skill lower bound threshold value. The time at which worker's skill reaches the lower bound level is represented by $t_{i,j}^{A'}$ (Figure 4-1).

As mentioned above, once a worker is transferred to another workstation, his (her) new skill begins to improve and if no transformation occurs it reaches the upper bound (S^{UB}) at $t_{i,j}^{A}$ (Figure 4-1). However, as the worker continues to learn the new skill, his (her) previously gained skill(s) decays as a result of the forgetting phenomenon (Figure 4-2). If the worker does not practice his (her) skill for an extended amount of time, it may gradually diminish until reaches the lower bound (S^{LB}) at $t_{i,j}^{A'}$ (Figure 4-1).

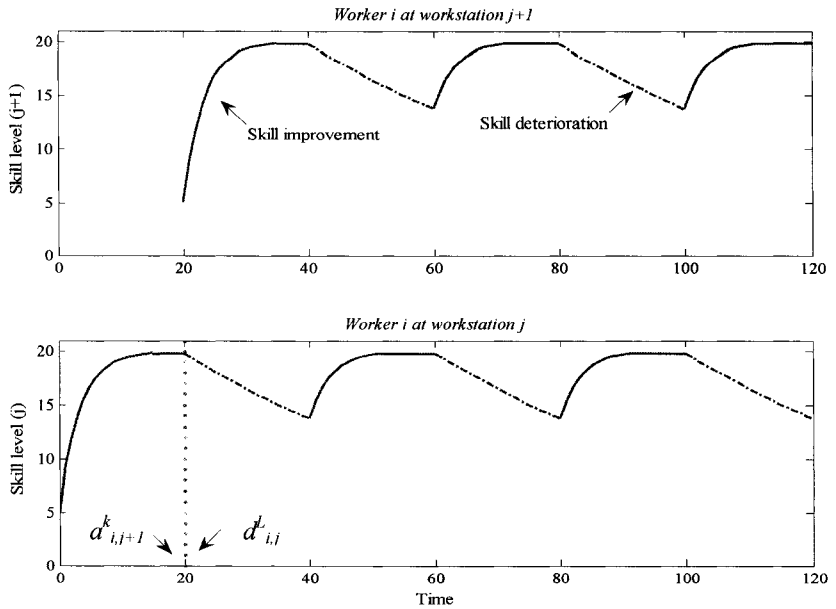


Figure 4-2. Worker *i* skill improvement and deterioration curves in consecutive assignments to two workstations

4.3.3 Motivation (boredom recovery) and boredom curves

The importance of emotions in the workplace is being recognized (Ashkanasy *et al.*, 2002). However, the study of boredom at work is still a neglected issue as pointed out by Fisher (1993) more than a decade ago (Game, 2007). Boredom at work is a common complaint among employees. It is defined as an undesirable transient state in which individuals feel an extreme lack of interest in their current activity. Boredom has been criticized for employee's absenteeism, accidents, performance variation, and lack of job satisfaction. It is usually accompanied by feelings of restlessness, irritability, and desire to escape or change the situation to a more interesting activity. Traditionally, factors causing boredom are assumed to be external such as repetitive or un-stimulating tasks. However, according to the recent studies (e.g., Game, 2007; Fisher, 1993; Drory, 1982) there are possible internal causes of boredom relating to individual personality. Moreover, studies have shown that the degrees of boredom reported by different individuals performing in the same repetitive working environment may vary significantly. It has been also shown that boredom could be highly situation-specific and is reversible when the situation changes to some extent (Farmer and Sundberg 1986).

Loss of task motivation might be conceptualized as boredom (Sawin and Scerbo, 1995). Likewise, motivation gain can be interpreted as boredom recovery. In this study, it is assumed that when an operator with a relatively low motivation level (caused by performing a repetitive task) is transferred to another workstation, his (her) motivation will gradually increase (boredom recovery). The increment in worker's motivation level continues at the new workplace until it reaches a maximum level. Worker's motivation level while performing a particular operation at any time during production horizon may depend on several factors. Such factors are worker's motivation level at the beginning of the assignment, the amount of time worker has been performing the same operation, and worker's motivation slope. Given individual differences in personality, the slope of motivation controls how fast or slow an employee may recover from the emotional state of boredom once the situation is changed. Therefore, mathematical expression for worker's motivation could be presented as follows

$$V_{i,j,t} = V^{\max} - \left(V^{\max} - V_{i,j-1,a_{i,j}^k}\right) e^{-\tau_i\left(t-a_{i,j}^k\right)} \tag{4.10}$$

where $V^{\max}$ is a theoretical maximum level of motivation, $V_{i,j-1,a_{i,j}^k}$ is the worker's motivation level in previous workstation at the transformation time, τ_i is the worker's motivation slope, and $a_{i,j}^k$ is the time at which worker i moves from workstation j-1 to j.

According to the above equation, worker's motivation reaches the maximum level, $V^{\max}$, at infinite time (*i.e.* $t \to \infty$,). The *achievable motivation* level or *motivation upper bound* is represented by V^{UB}. The following equation establishes a relationship between $V^{\max}$ and V^{UB}

$$V^{UB} = V^{\max} - \delta_1 \tag{4.11}$$

where δ_1 is the motivation upper bound threshold value. The time at which worker's motivation reaches the upper bound level is represented by $t_{i,j}^C$ (Figure 4-3).

Time at which work's motivation reaches the upper bound level, $t_{i,j}^C$, depends primarily on his (her) initial level of motivation and his (her) slope of motivation (boredom recovery). On the other hand, time at which worker's motivation begins

declining (job boredom), $t_{i,j}^{mdc}$, depend only on the time duration the worker spends at a workstation performing a specific operation repeatedly.

These two time points, $t_{i,j}^{C}$ and $t_{i,j}^{mdc}$, either coincide or occur at different times. The following two cases describe the two situations.

First, consider a case in which a worker with motivation level lower than or equal to $V^{Initial}$ transfers to another workstation. The worker's motivation at the new workstation gradually increases until it reaches the upper bound level at $t_{i,j}^{C}$. Following the time point $t_{i,j}^{C}$, if the worker continues to perform the same operation, his (her) motivation is assumed to decline as a result of boredom. In such cases, the time at which the worker's motivation reaches the upper bound coincides with the time at which it begins declining.

Another situation arises when the same worker moves to another workstation with a motivation level higher than $V^{Initial}$. Similar to the previous case, worker's motivation grows gradually until it reaches the upper bound level. However, given the worker's initial motivation level, in this case the worker's motivation is expected to reach the upper bound earlier than the one described in previous case. As a result, in such cases it is assumed that the worker's motivation remains at the upper bound level for a period of time before declining even though he (she) may still be performing the same operation. The length of the interval between $t_{i,j}^{C}$ and $t_{i,j}^{mdc}$ depends primarily on how close to the upper bound level at the time of transformation the worker's motivation is.

Hence, the expression for the time point $t_{i,j}^{mdc}$ could be driven as:

$$t_{i,j}^{mdc} = \max\left\{t_{i,j}^{C}, D_i^{V^{UB}} + a_{i,j}^{k}\right\} \tag{4.12}$$

where $D_i^{V^{UB}}$ is the duration time required for the motivation of worker *i* to reach the upper bound beginning from an initial level $V^{Initial}$ while performing a particular operation.

As soon as the time point $t_{i,j}^{mdc}$ is met, the worker's motivation level begins to decline and reaches the lower bound at $t_{i,j}^{C'}$ if no transformation occurs. The decrement in worker's motivation could be described by the following equation:

$$V_{i,j,t} = V^{\min} + (V_{i,j,t^{C}} - V^{\min})e^{-\theta_i\left(t - t_{i,j}^{mdc}\right)} \tag{4.13}$$

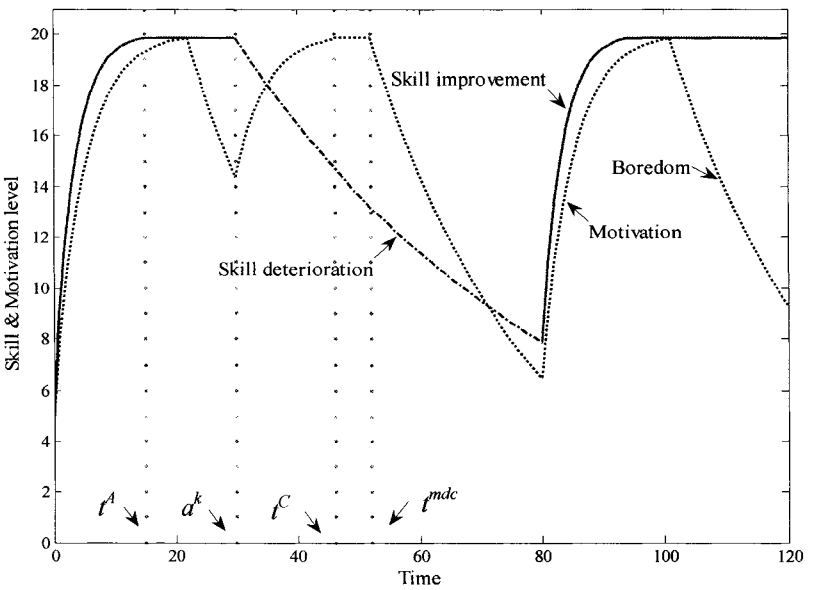


Figure 4-3. Worker skill and motivation curves at a workstation

where $V^{\min}$ is a theoretical minimum level of motivation, and θ_i is the slope of boredom.

As discussed earlier, studies have shown that not all individuals are equally likely to experience the same level of boredom. For instance, Drory (1982) showed that a younger, intelligent male, with relatively high tenure can easily become bored at work. O'Hanlon (1981) found that extroverts also have a lower threshold for boredom. Therefore, in equation (4.13) a personal slope of motivation or boredom recovery, i.e., θ_i, is assigned to each worker representing individual differences in developing boredom.

If the minimum level of motivation is considered to be zero, then the above equation can be simplified as

$$V_{i,j,t} = V_{i,j,t^C}\, e^{-\theta_i\left(t - t_{i,j}^{mdc}\right)}$$

According to equation (4.13), the motivation of worker *i* performing operation *j* reaches its minimum level, $V^{\min}$, at infinite time(i.e., $t \to \infty$). The *achievable minimum level of motivation* which is also called *motivation lower bound* is represented by V^{LB}. The relationship between $V^{\min}$ and V^{LB} can be expressed as

$$V^{LB} = V^{\min} + \delta_2 \qquad (4.14)$$

where δ_2 is motivation lower bound threshold value. The time at which worker's motivation reaches the lower bound level is represented by $t_{i,j}^{C'}$ (Figure 4-3). In Figure 4-3, the solid and dash blue lines represent the worker's skill improvement and deterioration curves. The dash green lines represent the variation in worker's motivation respectively.

4.4 Mathematical model

4.4.1 Objective function

In this study, the objective function sums the (possible) Lost Production Time (LPT) for each operation performed by one or more workers over the entire production horizon:

$$\sum_{i=1}\sum_{j=1}\sum_{t=1}\left[\left(\frac{S^{UB}-S_{i,j,t}}{S^{UB}}\right)\alpha+\left(\frac{V^{UB}-V_{i,j,t}}{V^{UB}}\right)(1-\alpha)\right]x_{i,j,t} \tag{4.15}$$

The first term of the objective function calculates the portion of the LPT resulted from the worker's lack of skill and the second term calculates the lost production time caused by worker's lack of motivation.

4.4.2 Time points and duration time equations

The first time point to consider is $t_{i,j}^{A}$, the time at which the skill of worker *i* reaches the upper bound as he (she) performs operation *j*. The appropriate expression for $t_{i,j}^{A}$ can be derived by substituting $S_{i,j,t}$ and t by S_{i,j,t^A} and $t_{i,j}^{A}$ in equation (4.3) respectively,

$$S_{i,j,t^A}=S^{UB}=S^{\max}-\left(S^{\max}-S_{i,j,a^k}^{Rem}\right)\ e^{\beta_i\left(t_{i,j}^{A}-a_{i,j}^{k}\right)} \tag{4.16}$$

Furthermore, considering equation (4.5), the above equation can be written as

$$S^{\max}-\varepsilon_1=S^{\max}-\left(S^{\max}-S_{i,j,a^k}^{Rem}\right)\ e^{\beta_i\left(t_{i,j}^{A}-a_{i,j}^{k}\right)}, \tag{4.17}$$

solving the above equation for $t_{i,j}^{A}$ gives

$$t_{i,j}^{A} = \frac{\ln\left(\frac{\varepsilon_1}{S^{\max} - S_{i,j,a^k}^{\mathrm{Re}m}}\right)}{\beta_i} + a_{i,j}^{k} \tag{4.18}$$

Similarly, the expression for $t_{i,j}^{A'}$, i.e., the time at which the worker's skill reaches the lower bound, could be obtained by first substituting for $S_{i,j,t}$ and t in equation (4.6),

$$S_{i,j,t^{A'}} = S^{LB} = S^{\min} + (S_{i,j,d^L} - S^{\min}) e^{\gamma_i \left(t_{i,j}^{A'} - d_{i,j}^{L}\right)} \tag{3.19}$$

and solving equation (4.19) for $t_{i,j}^{A'}$ with consideration of equation (4.9):

$$t_{i,j}^{A'} = \frac{\ln\left(\frac{\varepsilon_2}{S_{i,j,d^L} - S^{\min}}\right)}{\gamma_i} + d_{i,j}^{L} \tag{4.20}$$

Similarly, using equations (4.10) and (4.13) and the following two expressions

$$V^{UB} = V^{\max} - \delta_1$$

$$V^{LB} = V^{\min} + \delta_2$$

the corresponding formulas for time point $t_{i,j}^{C}$ and $t_{i,j}^{C'}$ are determined as

$$t_{i,j}^{C} = \frac{-\ln\left(\frac{\delta_1}{V^{\max} - V_{i,j-1,a_{i,j}^k}}\right)}{\tau_i} + a_{i,j}^{k} \tag{4.21}$$

and

$$t_{i,j}^{C'} = \frac{-\ln\left(\frac{\delta_2}{V_{i,j,t^C} - V^{\min}}\right)}{\theta_i} + t_{i,j}^{mdc} \tag{4.22}$$

To obtain an expression for $D_i^{V^{UB}}$, i.e., the amount of time required for worker's motivation to reach the upper bound from an initial level of $V^{Initial}$, assume that operation j is the first operation performed by worker i in current production horizon. As a result, $a_{i,j}^{k} = 0$ and $V_{i,j-1,a_{i,j}^k} = V_{i,0,0} = V^{Initial}$. Substituting the $a_{i,j}^{k}$ and $V_{i,j-1,a_{i,j}^k}$ values into equation (4.10) gives

$$V_{i,j,t^C} = V^{UB} = V^{\max} - \left(V^{\max} - V^{Initial}\right) e^{-\tau_i (t-0)} \tag{4.23}$$

Combining equations (4.11) and (4.23) leads to

$$t = \frac{-\ln\left(\frac{\delta_1}{V^{\max} - V^{Initial}}\right)}{\tau_i} \tag{4.24}$$

or

$$t = D_i^{V^{UB}} = \frac{-\ln\left(\frac{\delta_1}{V^{\max} - V^{Initial}}\right)}{\tau_i} \tag{4.25}$$

4.4.3 Model formulation

Based on the above analysis, a mathematical programming model is formulated for the job rotation problem in cellular manufacturing systems as follows:

$$\text{Min} \sum_{i=1}\sum_{j=1}\sum_{t=1}\left[\left(\frac{S^{UB} - S_{i,j,t}}{S^{UB}}\right)\alpha + \left(\frac{V^{UB} - V_{i,j,t}}{V^{UB}}\right)(1-\alpha)\right]x_{i,j,t}$$

Subject to:

The classical job rotation constraints

$$\sum_{i=1}^{m} x_{i,j,t} = 1 \qquad \forall j,t \in (\mathbf{J},\mathbf{T}) \tag{4.26}$$

$$\sum_{j=1}^{m} x_{i,j,t} = 1 \qquad \forall i,t \in (\mathbf{I},\mathbf{T}) \tag{4.27}$$

The relationship between the two decision variables $x_{i,j,t}$ (independent variable) and $a_{i,j}^k$ (dependent variable)

$$a_{i,j}^k = \left(d_{i,j}^L + \sum_{l=d_{i,j}^L}^{t}\sum_{\substack{r=1 \\ r\neq j}}^{m} x_{i,r,l}\right)x_{i,j,t} \qquad \forall i,j,t \in (\mathbf{I},\mathbf{J},\mathbf{T}), k \in \{1,2,3,...,n\} \tag{4.28}$$

The relationship between $a_{i,j}^k$ and $d_{i,j}^L$

$$d_{i,j}^L - a_{i,j+1}^k = 0 \qquad \forall i,j,t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \mid a_{i,j}^k \neq 0, k \in \{1,2,3,...,n\} \tag{4.29}$$

The skill levels of workers at each time period t

$$S_{i,j,t} \leq S^{UB} \qquad \forall i,j,t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \tag{4.30}$$

$$S_{i,j,t} \geq S^{UB} v_{i,j}^1 \qquad \forall i,j,t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \tag{4.31}$$

$$S_{i,j,t} \le S^{\max} - \left(S^{\max} - S^{Rem}_{i,j,a^k}\right) e^{\beta_i\left(t-a^k_{i,j}\right)} + M_1 v^1_{i,j} \qquad \forall i,j,t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \tag{4.32}$$

$$S_{i,j,t} \ge S^{\max} - \left(S^{\max} - S^{Rem}_{i,j,a^k}\right) e^{\beta_i\left(t-a^k_{i,j}\right)} - M_1 v^1_{i,j} \qquad \forall i,j,t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \tag{4.33}$$

$$S^{Rem}_{i,j,a^k} \le S^{LB} + M_2\left(1-v^2_{i,j}\right) \qquad \forall i,j,t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \tag{4.34}$$

$$S^{Rem}_{i,j,a^k} \ge S^{LB} - M_2\left(1-v^2_{i,j}\right) \qquad \forall i,j,t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \tag{4.35}$$

$$S^{Rem}_{i,j,a^k} \le S^{\min} + (S_{i,j,d^L} - S^{\min}) e^{\gamma_i\left(a^k_{i,j}-d^L_{i,j}\right)} + M_3\left(1-v^3_{i,j}\right) \qquad \forall i,j,t \in (\mathrm{I},\mathrm{J},\mathrm{T}) \tag{4.36}$$

$$S^{Rem}_{i,j,a^k} \ge S^{\min} + (S_{i,j,d^L} - S^{\min}) e^{\gamma_i\left(a^k_{i,j}-d^L_{i,j}\right)} - M_3\left(1-v^3_{i,j}\right) \qquad \forall i,j,t \in (\mathrm{I},\mathrm{J},\mathrm{T}) \tag{4.37}$$

$$S^{Rem}_{i,j,a^k} \le S^{Initial} + M_4\left(1-v^4_{i,j}\right) \qquad \forall i,j,t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \tag{4.38}$$

$$S^{Rem}_{i,j,a^k} \ge S^{Initial} + M_4\left(1-v^4_{i,j}\right) \qquad \forall i,j,t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \tag{4.39}$$

The motivation levels of workers for each time period t

$$V_{i,j,t} \le V^{UB} \qquad \forall i,j,t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \tag{4.40}$$

$$V_{i,j,t} \ge V^{UB} u^1_{i,j} \qquad \forall i,j,t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \tag{4.41}$$

$$V_{i,j,t} \le V^{\max} - \left(V^{\max} - V_{i,j-1,a^k_{i,j}}\right) e^{-\tau_i\left(t-a^k_{i,j}\right)} + \left(1-u^2_{i,j}\right)M_5 \quad \forall i,j,t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \tag{4.42}$$

$$V_{i,j,t} \ge V^{\max} - \left(V^{\max} - V_{i,j-1,a^k_{i,j}}\right) e^{-\tau_i\left(t-a^k_{i,j}\right)} - \left(1-u^2_{i,j}\right)M_5 \quad \forall i,j,t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \tag{4.43}$$

$$V_{i,j,t} \le V^{LB} + \left(1-u^3_{i,j}\right)M_6 \qquad \forall i,j,t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \tag{4.44}$$

$$V_{i,j,t} \ge V^{LB} \qquad \forall i,j,t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \tag{4.45}$$

$$V_{i,j,t} \le V^{\min} + (V_{i,j,t^C} - V^{\min})\ e^{-\theta_i\left(t-t^{mdc}_{i,j}\right)} + \left(1-u^4_{i,j}\right)M_7 \qquad \forall i,j,t \in (\mathrm{I},\mathrm{J},\mathrm{T}) \tag{4.46}$$

$$V_{i,j,t} \ge V^{\min} + (V_{i,j,t^C} - V^{\min})\ e^{-\theta_i\left(t-t^{mdc}_{i,j}\right)} - \left(1-u^4_{i,j}\right)M_7 \qquad \forall i,j,t \in (\mathrm{I},\mathrm{J},\mathrm{T}) \tag{4.47}$$

$$V_{i,0,a^k_{i,j}} = V^{Initial} \qquad \forall i, a^k_{i,j} \in (\mathbf{I},\mathbf{T}) \tag{4.48}$$

Constraints to ensure that only a single $u^l_{i,j}$ equals one

$$\sum_{l=1}^{4} u^l_{i,j} = 1 \qquad \forall i,j \in (\mathbf{I},\mathbf{J}), l = 1,2,3,4 \tag{4.49}$$

Constraints to ensure that only a single $v^l_{i,j}$ equals one

$$\sum_{l=1}^{4} v^l_{i,j} = 1 \qquad \forall i,j \in (\mathbf{I},\mathbf{J}), l = 1,2,3,4 \tag{4.50}$$

The computed time points

$$t^A_{i,j} = \frac{\ln\left(\dfrac{\varepsilon}{S^{\max} - S^{Rem}_{i,j,a^k}}\right)}{\beta_i} + a^k_{i,j} \qquad \forall i,j,t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \tag{4.51}$$

$$t_{i,j}^{A'} = \frac{\ln\left(\frac{\varepsilon}{S_{i,j,d^L} - S^{\min}}\right)}{\gamma_i} + d_{i,j}^{L} \qquad \forall i,j,t \in (\mathrm{I,J,T}) \tag{4.52}$$

$$t_{i,j}^{C} = \frac{-\ln\left(\frac{\delta}{V^{\max} - V_{i,j-1,a_{i,j}^k}}\right)}{\tau_i} + a_{i,j}^{k} \qquad \forall i,j,t \in (\mathbf{I,J,T}) \tag{4.53}$$

$$t_{i,j}^{C'} = \frac{-\ln\left(\frac{\delta}{V_{i,j,t^C} - V^{\min}}\right)}{\theta_i} + t_{i,j}^{mdc} \qquad \forall i,j,t \in (\mathrm{I,J,T}) \tag{4.54}$$

$$t_{i,j}^{mdc} = \mathbf{max}\left\{ t_{i,j}^{C}, D_i^{V^{UB}} + a_{i,j}^{k} \right\} \qquad \forall i,j,t \in (\mathbf{I,J,T}) \tag{4.55}$$

The time required for motivation of worker i to reach the upper bound starting from an initial level $V^{Initial}$

$$D_i^{V^{UB}} = \frac{-\ln\left(\frac{\delta}{V^{\max} - V^{Initial}}\right)}{\tau_i} \qquad \forall i \in (\mathbf{I}) \tag{4.56}$$

Minimum time interval between the two assignments to the same workstation

$$a_{i,j}^{k} - d_{i,j}^{L} \geq \Omega_{i,j} \qquad \forall i,j,t \in (\mathbf{I,J,T}) \tag{4.57}$$

Types of variables

$$x_{i,j,t}, v_{i,j}^{l}, u_{i,j}^{l} \in \{0,1\} \quad \forall i,j,t \in (\mathbf{I,J,T}), l = 1,2,3,4 \tag{4.58}$$

$$t_{i,j}^{A}, t_{i,j}^{A'}, a_{i,j}^{k}, t_{i,j}^{C}, t_{i,j}^{C'}, d_{i,j}^{L}, t_{i,j}^{mdc}, D_i^{V^{UB}} \geq 0 \quad \mathbf{Integer} \quad \forall i,j \tag{4.59}$$

$$\mathrm{S_{i,j,t}}, \mathrm{V_{i,j,t}} \in \{\mathrm{R}\}(\text{real number}) \tag{4.60}$$

The objective of the model is to minimize the total lost production time caused by the worker's lack of skill and/or motivation during production horizon. The first two sets of constraints, (4.26) and (4.27), ensure respectively that each operation is performed by a single worker across each of the time units and each worker performs only one operation in each time unit. Constraint (4.28) establishes the relationship between the two decision variables $x_{i,j,t}$ (independent) and $a_{i,j}^{k}$ (dependent). $a_{i,j}^{k}$, time at which worker i begins (or has begun) operating at workstation j for the k^{th} time, is calculated as the sum of $d_{i,j}^{L}$, the time that worker i last visited workstation j, and the total time that the

worker has spent at other workstations prior to being reassigned to workstation *j*. Constraint (4.29) describes the relationship between $a_{i,j}^{k}$ and $d_{i,j}^{L}$. Constraints (4.30) to (4.39) calculate the skill levels of workers at each time period *t*. If time *t* is equal to or greater than the time point at which the skill of worker *i* reaches the upper bound, i.e., $t \geq t_{i,j}^{A}$, $v_{i,j}^{1} = 1$ and constraints (4.30) and (4.31) are binding. Otherwise, depending on whether or not the worker has been performing operation *j* in the past, the following cases are examined. Case (a) addresses a situation in which worker *i* is reassigned to workstation *j* when his (her) skill (*j*) remnant has already reached the minimum level. In Case (b), we also examines a situation in which the worker is reassigned to the workstation. However, in this case his (her) level of skill remnant is above the minimum level. Case (c) deals with the situation in which worker *i* is assigned to workstation *j* for the first time. In this case the remnant of worker's skill will be equal to worker's initial skill level.

Case (a): if time *t* is greater than or equal to $t_{i,j}^{A'}$, the time at which worker *i* remnant of skill *j* reaches its minimum level, $v_{i,j}^{2} = 1$ and constraints (4.32), (4.33), (4.34), and (4.35) are binding. Case (b): if time *t* is less than $t_{i,j}^{A'}$, $v_{i,j}^{3} = 1$ and constraints (4.32), (4.33), (4.36), and (4.37) are binding. Case(c): if worker *i* is assigned to workstation *j* for the first time, $d_{i,j}^{L} = 0$ (time of the last departure from workstation *j* is considered to be zero), $v_{i,j}^{4} = 1$ and constraints (4.32), (4.33), (4.38), and (39) are applied.

Constraints (4.40) to (4.48) calculate the motivation levels of workers for each time period *t*. If time *t* is greater than or equal to $t_{i,j}^{C}$ (time at which the motivation level of worker *i* in performing operation *j* reaches the upper bound), and less than $t_{i,j}^{mdc}$ (time at which the motivation of worker *i* in performing operation *j* begins declining), then $u_{i,j}^{1} = 1$ and constraints (4.40) and (4.41) are binding. If time *t* is less than $t_{i,j}^{C}$, i.e., the motivation of worker *i* has not reached the upper bound, $u_{i,j}^{2} = 1$ and constraints (4.42) and (4.43) are applied. If time *t* is greater than or equal to the time point $t_{i,j}^{C'}$ (time at which the motivation level of worker *i* in performing operation *j* reaches the lower bound),

constraints (4.44) and (4.45) are binding. If time t is greater than or equal to the time point $t_{i,j}^{mdc}$, i.e., time at which the motivation of worker i begins declining, constraints (4.46) and (4.47) are binding.

Constraints (4.51) to (4.55) compute the time points $t_{i,j}^{A}$ (time at which the skill of worker i in performing operation j reaches the upper bound), $t_{i,j}^{A'}$(time at which the remnant of skill j gained by worker i reaches the lower bound), $t_{i,j}^{C}$, $t_{i,j}^{C'}$, and $t_{i,j}^{mdc}$ for worker i performing operation j at any time t during the current production horizon. Constraint (4.56) calculates the duration time required for motivation of worker i to reach the upper bound starting from an initial level $V^{Initial}$. In some situations a managerial decision may prevent the reallocation of a worker to the same workstation for a certain amount of time (e.g., because of harsh working conditions). Constraint (4.57) ensures that minimum interval time between the time at which worker i has last departed from workstation j and the time that he (she) revisits the same workstation is met. To illustrate the application of the proposed model, the following numerical example is provided.

4.5 Numerical example

Consider a manufacturing cell in which two workstations (i.e., machines) are dedicated to the production of a family of parts. Each workstation is operated by a single worker during the planning horizon. Learning and forgetting curve coefficients associated with worker 1 and worker 2 (r_1, r_2) are assumed to be respectively 0.850 and 0.987. It is also assumed that worker 1's motivation and boredom slopes be respectively 0.20 and 0.04. Worker 2's learning and forgetting coefficients, motivation slope, and boredom slope are assumed to be respectively 0.920, 0.950, 0.17, and 0.03. Furthermore, it is assumed that the maximum and minimum level of both worker's skill and motivation are respectively 20 and zero, respectively. The initial level of both skill and motivation is considered to be five. The threshold values for both skill and motivation is set to 0.20 and it is assumed that the standard time to perform each operation by a skilled and motivated worker is one time unit. The minimum interval between the two consecutive assignments to the same workstation is assumed to be 5. For a planning horizon of 120 time units, the following scenarios are investigated.

***Case 1:* No rotation**

Worker 1 will perform operation 1 and worker 2 is assigned to operation 2 during the entire production horizon. According to equation (4.49), the skill of worker 1 and worker 2 will reach the upper bound at times 18 and 24, respectively. The workers skill levels will stay at the upper bound for the remaining time of the current production horizon. Using equation (4.51), the motivation level of worker 1 and worker 2 will reach the upper bound (19.8) respectively at times 22 and 25. As workers continue to perform the same operations, their motivation gradually declines for the rest of the production horizon.

Case 2: Swap the workers at time 41

Workers 1 and 2 will perform respectively operations 1 and 2 for 41 time units. At the end of time 41, the workers are swapped such that for the rest of the production horizon worker 1 will perform operation 2 and worker 2 will do the operation 1.

Case 3: swap the workers at times 41 and 49

In this case, the workers are swapped twice during production horizon. First, at the beginning of time 42 the workers are exchanged (worker 1 is assigned to operation 2 and worker 2 is assigned to operation 1). At the end of time 49 they will return to their previous workstations.

Case 4: Swap the workers every 8 time units beginning at time 33

The workers will first perform the assigned operations for 32 units of time. Then for the rest of the production horizon, they are swapped after every 8 time units. Table 4-1 summarizes the total delays over the production horizon caused by the lack of skill and motivation for each of the above scenarios. The value of the objective function for each scenario is compared with the others considering different weight values of α.

Table 4-1. Computational results for numerical example

	Total Lost Production Time				
	α				
	0.5	*0.6*	*0.7*	*0.8*	*0.9*
Case 1	75.6	61.7	47.8	33.9	20.1
Case 2	47.7	40.7	33.6	26.6	19.6
Case 3	41.9	36.4	30.7	25.2	19.6
Case 4	23.6	25.9	28.4	30.7	33.1

In case 1, depending on the magnitude of the objective function weight value, α, the total lost production time (LPT) ranges from 75.6 to 20.1 unites. When an equal weight of 0.5 is considered for both criteria, the result shows that the total LPT significantly reduces when a rotation plan partially (cases 2 and 3) or more completely (case 4) is implemented. However, as the skill improvement weight increases, the effect of implementing a rotation plan on reducing total LPT diminishes. More specifically, the result pertaining to case 4 ($\alpha = 0.9$) clearly indicates that the implementation of a rotation plan may even increase the total LPT (i.e., may have a negative effect on productivity) when the primary objective is the worker skill improvement. Figure 4-4 compares the total lost production time associated with four cases described above when the objective function weight value, α, is 0.5.

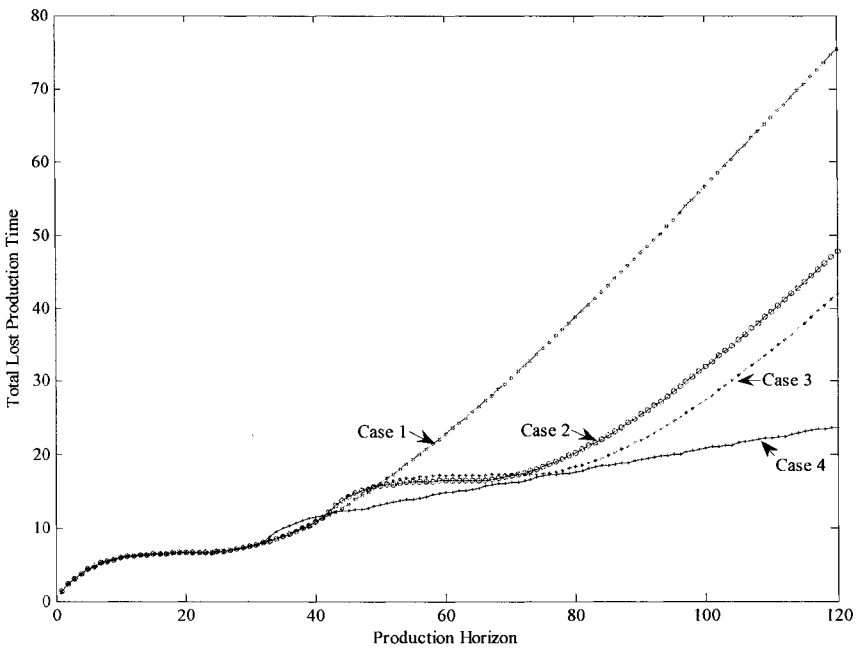
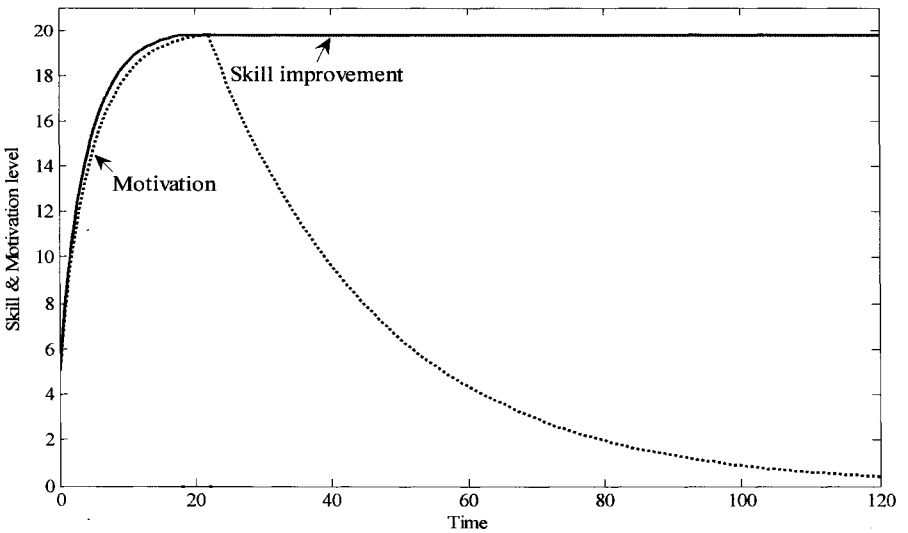
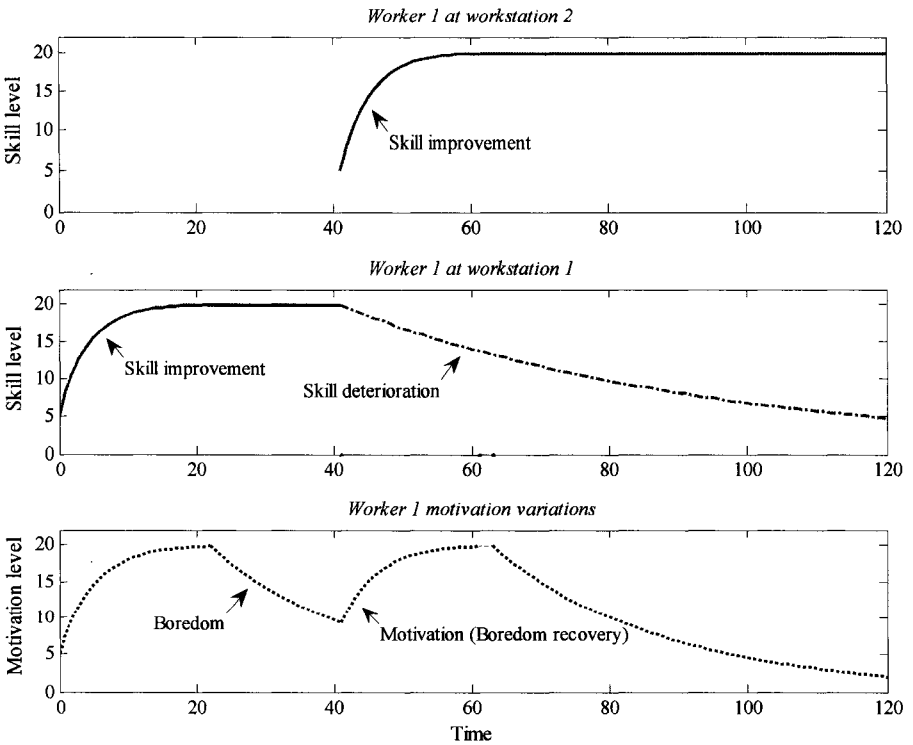


Figure 4-4. Comparison of the total lost production times over the production horizon

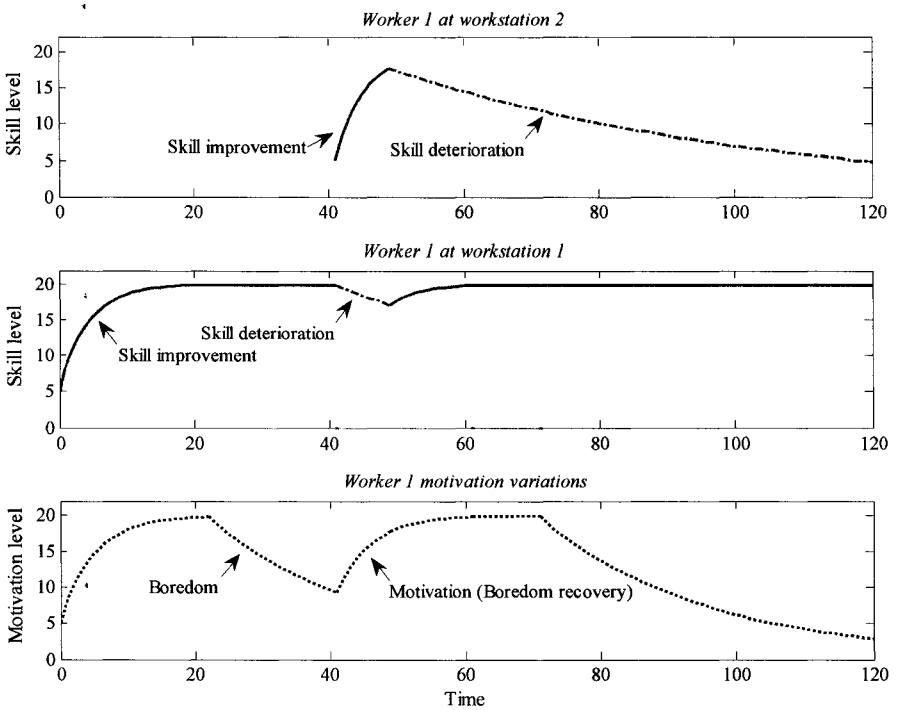
The skill and motivation variations of worker 1 in the above four cases have been depicted in Figure 4-5.



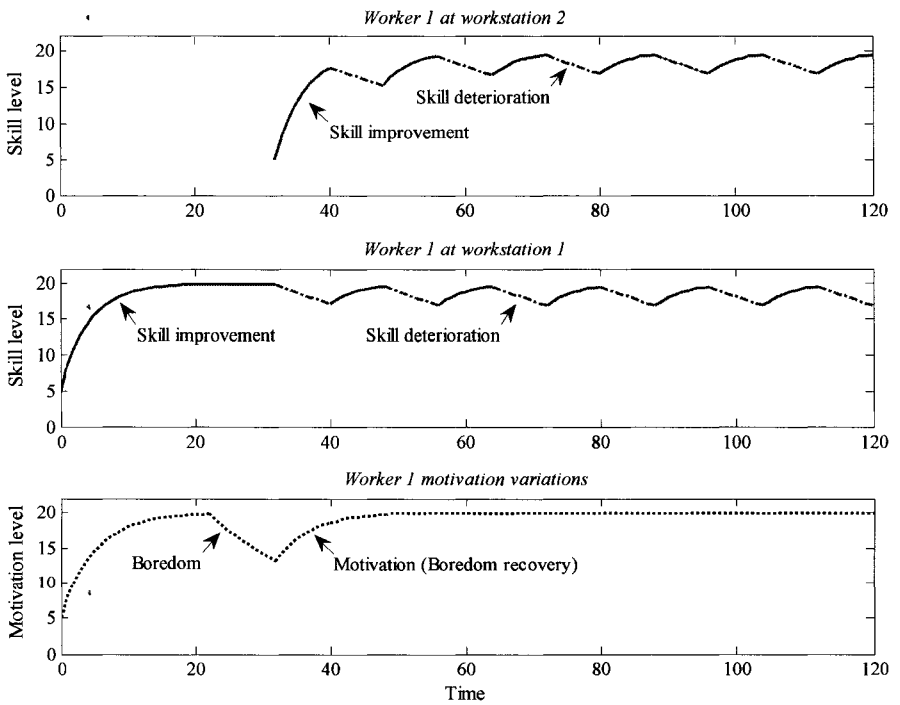
(a) Case 1



(b) Case 2



(c) Case 3



(d) Case 4

Figure 4-5. Graphical illustration of the numerical example

In Figure 4-5, the solid and dash blue lines represent the worker's skill improvement and deterioration curves respectively. The dash green lines represent the variation in worker's motivation.

4.6 Linear version of the proposed model

In the above model, the worker's motivation was described by exponential functions. To reduce the complexity, a linear version of the model is presented in this subsection. However, due to the similarities between the two linear and nonlinear models, only a brief discussion about the equations will be provided here.

4.6.1 Skill improvement and deterioration lines

In this version of the model it is assumed that the workers' skill grows linearly as they continue performing the same task over time. The improvement in a worker's skill depends on the worker's initial level of skill, learning coefficient, and duration of the assignment. Therefore, the worker's skill improvement could be described as follow

$$S_{i,j,t} = S^{Rem}_{i,j,a^k} + r_i\left(t - a^k_{i,j}\right) \tag{4.61}$$

In the above equations $S_{i,j,t}$ is the skill level of worker i performing operation j at time t, S^{Rem}_{i,j,a^k} is the worker's remnant of skill at the beginning of the assignment, r_i is the worker's learning coefficient, and $a^k_{i,j}$ is the time at which the k^{th} assignment of the worker begins at workstation j (see Figure 4-6).

Once a worker is assigned to a new work station, his/her corresponding skill begins to grow according to the equation (4.61). This trend continues until the skill reaches the maximum level of S^{max} at time $t^A_{i,j}$. The time at which worker's skill reaches the maximum level is calculated by the following equation

$$t^A = \frac{S^{max} - S^{Rem}_{i,j,a^k}}{r_i} + a^k_{i,j} \tag{4.62}$$

However, because of the forgetting phenomenon his/her previous skills begin to decay as the worker starts learning new skills. If the worker does not practice his/her previously learned skills for an extended period of time, they will gradually diminish and

eventually reach the minimum level of $S^{\min}$. The remnant of worker's skill (e.g., skill j) at time t depends on the worker's skill at the time he/she last departed from workstation j, the forgetting coefficient, and the interval time between time t and the last departure time of the worker from workstation j. Hence, the deterioration of worker's skill is described according to the following formula

$$S_{i,j,t}^{Rem} = S_{i,j,d^L} + f_i\left(t - d_{i,j}^L\right) \tag{4.63}$$

where S_{i,j,d^L} is the worker's skill level immediately after leaving (last time) workstation j, f_i is the worker's forgetting coefficient, and $d_{i,j}^L$ is the worker's last departure time from workstation j (see Figure 4-6).

The time at which worker's skill reaches the minimum level is represented by $t_{i,j}^{A'}$ and could be calculated according to the following formula

$$t^{A'} = \frac{S^{\min} - S_{i,j,d^L}}{f_i} + d_{i,j}^L \tag{4.64}$$

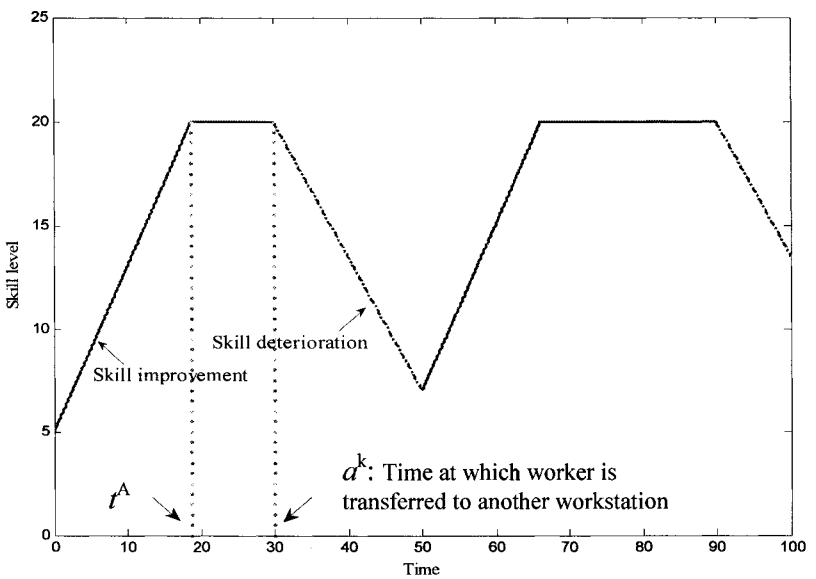


Figure 4-6. Skill improvement and deterioration lines

4.6.2 Motivation and boredom lines

Similar to the worker's skill, variation in worker's motivation is also assumed to change linearly over time. The worker's motivation level at any time during production horizon depends on initial state of the worker's motivation at the time he/she begins operating in a new workstation, the amount of time spent at the workstation, and his/her motivation coefficient. The mathematical formula to describe the worker's motivation is presented as follows

$$V_{i,j,t} = V_{i,j-1,a_{i,j}^k} + h_i\left(t - a_{i,j}^k\right) \tag{4.65}$$

where $V_{i,j,t}$ is the motivation of worker i at time t while performing operation j, $V_{i,j-1,a_{i,j}^k}$ is the worker's motivation at the end of the previous assignment, $a_{i,j}^k$ is the time at which the k^{th} assignment of the worker begins at workstation j, and h_i is the worker's motivation coefficient.

It is further assumed that once a worker is assigned to another workstation, his/her motivation will increase and eventually reaches a maximum level of $V^{\max}$ at a certain time represented by $t_{i,j}^C$. The time at which worker's motivation reaches the maximum level is calculated by

$$t^C = \frac{V^{\max} - V_{i,j-1,a^k}}{h_i} + a_{i,j}^k \tag{4.66}$$

If the worker continues performing the same operation however, his/her motivation may begin declining after time $t_{i,j}^C$ (see Figure 4-7).

The decrement in worker's motivation is described by the following equation

$$V_{i,j,t} = V_{i,j,t^C} + g_i\left(t - t_{i,j}^{mdc}\right) \tag{4.67}$$

In the above equation g_i is the worker's coefficient of boredom and $t_{i,j}^{mdc}$ is the time at which worker's motivation starts declining.

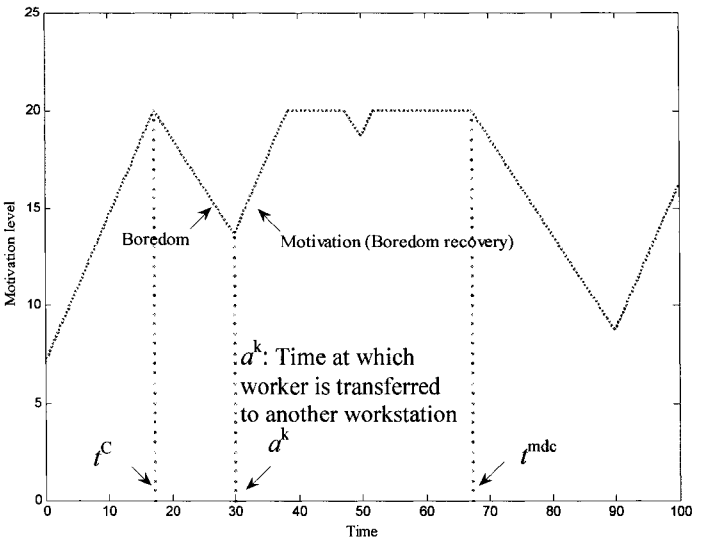


Figure 4-7. Motivation and boredom lines

The starting time at which worker's motivation begins declining (boredom development) depends primarily on the duration that the worker has spent at a particular workstation, j. Depending on the worker's motivation level at the time of transformation, boredom development begins either right after the time worker's motivation reaches its maximum level or following the duration required for the motivation of the worker to reach the maximum level (beginning from initial motivation) plus the transformation time, i.e.,

$$t_{i,j}^{mdc} = \mathbf{max}\left\{t_{i,j}^{C}, D_i^{V^{\max}} + a_{i,j}^{k}\right\} \tag{4.68}$$

where

$$D_i^{V^{\max}} = \frac{V^{\max} - V^{Initial}}{h_i} \tag{4.69}$$

$D_i^{V^{\max}}$ is the time required for the motivation of worker i to reach the maximum level beginning from an initial level $V^{Initial}$.

As soon as the time point $t_{i,j}^{mdc}$ is reached and if the worker continues to perform the same operation, his/her motivation begins declining. If the worker is not transferred to another workstation, it is assumed that his/her motivation continues to deteriorate until it

reaches the minimum level of $V^{\min}$ at $t_{i,j}^{C'}$. The expression to calculate the time point $t_{i,j}^{C'}$ is given as follows

$$t^{C'} = \frac{V^{\min} - V_{i,j,t^C}}{g_i} + t_{i,j}^{mdc} \tag{4.70}$$

An illustration of combined worker's skill and motivation variation is presented in Figure 4-8.

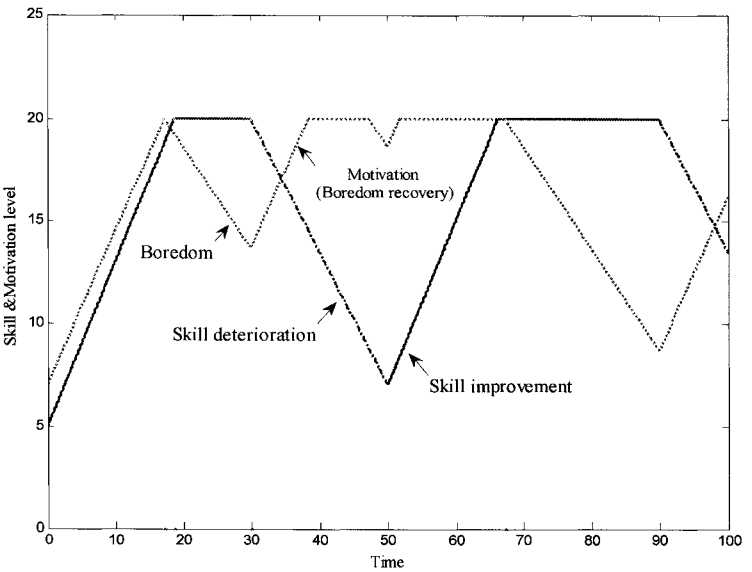


Figure 4-8. Worker's skill and motivation lines

4.6.3 Linear mathematical programming model

With the above equations, the linear version of the mathematical programming model for job rotation problem can be expressed as follows

$$\text{Min} \sum_{i=1}\sum_{j=1}\sum_{t=1}\left[\left(\frac{S^{UB} - S_{i,j,t}}{S^{UB}}\right)\alpha + \left(\frac{V^{UB} - V_{i,j,t}}{V^{UB}}\right)(1-\alpha)\right]x_{i,j,t} \tag{4.71}$$

Subject to:

The classical job rotation constraints

$$\sum_{i=1}^{m} x_{i,j,t} = 1 \qquad \forall j,t \in (\mathbf{J},\mathbf{T}) \tag{4.72}$$

(4.73)

$$\sum_{j=1}^{m} x_{i,j,t} = 1 \qquad \forall i,t \in (\mathbf{I},\mathbf{T})$$

The relationship between the two decision variables $x_{i,j,t}$ (independent variable) and $a_{i,j}^{k}$ (dependent variable)

$$a_{i,j}^{k} = \left(d_{i,j}^{L} + \sum_{l=d_{i,j}^{L}}^{t} \sum_{\substack{r=1 \\ r \neq j}}^{m} x_{i,r,l} \right) x_{i,j,t} \qquad \forall i,j,t \in (\mathbf{I},\mathbf{J},\mathbf{T}), k \in \{1,2,3,...,n\} \tag{4.74}$$

The relationship between $a_{i,j}^{k}$ and $d_{i,j}^{L}$

$$d_{i,j}^{L} - a_{i,j+1}^{k} = 0 \qquad \forall i,j,t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \middle| a_{i,j}^{k} \neq 0, k \in \{1,2,3,...,n\} \tag{4.75}$$

The skill levels of workers at each time period t

$$S_{i,j,t} \leq S^{\max} \qquad \forall i,j,t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \tag{4.76}$$

$$S_{i,j,t} \geq S^{\max} v_{i,j}^{1} \qquad \forall i,j,t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \tag{4.77}$$

$$S_{i,j,t} \leq S_{i,j,a^k}^{Rem} + r_i \left(t - a_{i,j}^{k}\right) + M_1 v_{i,j}^{1} \qquad \forall i,j,t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \tag{4.78}$$

$$S_{i,j,t} \geq S_{i,j,a^k}^{Rem} + r_i \left(t - a_{i,j}^{k}\right) - M_1 v_{i,j}^{1} \qquad \forall i,j,t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \tag{4.79}$$

$$S_{i,j,a^k}^{Rem} \leq S^{\min} + M_2 \left(1 - v_{i,j}^{2}\right) \qquad \forall i,j,t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \tag{4.80}$$

$$S_{i,j,a^k}^{Rem} \geq S^{\min} - M_2 \left(1 - v_{i,j}^{2}\right) \qquad \forall i,j,t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \tag{4.81}$$

$$S_{i,j,a^k}^{Rem} \leq S_{i,j,d^L} + f_i \left(a_{i,j}^{k} - d_{i,j}^{L}\right) + M_3 \left(1 - v_{i,j}^{3}\right) \qquad \forall i,j,t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \tag{4.82}$$

$$S_{i,j,a^k}^{Rem} \geq S_{i,j,d^L} + f_i \left(a_{i,j}^{k} - d_{i,j}^{L}\right) + M_3 \left(1 - v_{i,j}^{3}\right) \qquad \forall i,j,t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \tag{4.83}$$

$$S_{i,j,a^k}^{Rem} \leq S^{Initial} + M_4 \left(1 - v_{i,j}^{4}\right) \qquad \forall i,j,t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \tag{4.84}$$

$$S_{i,j,a^k}^{Rem} \geq S^{Initial} + M_4 \left(1 - v_{i,j}^{4}\right) \qquad \forall i,j,t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \tag{4.85}$$

The motivation levels of workers for each time period t

$$V_{i,j,t} \leq V^{\max} \qquad \forall i,j,t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \tag{4.86}$$

$$V_{i,j,t} \geq V^{\max} u_{i,j}^{1} \qquad \forall i,j,t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \tag{4.87}$$

$$V_{i,j,t} \leq V_{i,j-1,a_{i,j}^k} + h_i \left(t - a_{i,j}^{k}\right) + M_5 \left(1 - u_{i,j}^{2}\right) \qquad \forall i,j,t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \tag{4.88}$$

$$V_{i,j,t} \geq V_{i,j-1,a_{i,j}^k} + h_i \left(t - a_{i,j}^{k}\right) + M_5 \left(1 - u_{i,j}^{2}\right) \qquad \forall i,j,t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \tag{4.89}$$

$$V_{i,j,t} \leq V^{\min} + M_6 \left(1 - u_{i,j}^{3}\right) \qquad \forall i,j,t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \tag{4.90}$$

$$V_{i,j,t} \geq V^{\min} \qquad \forall i,j,t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \tag{4.91}$$

$$V_{i,j,t} \leq V_{i,j,t^C} + g_i \left(t - t_{i,j}^{mdc}\right) + M_7 \left(1 - u_{i,j}^{4}\right) \qquad \forall i,j,t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \tag{4.92}$$

$$V_{i,j,t} \geq V_{i,j,t^C} + g_i \left(t - t_{i,j}^{mdc}\right) + M_7 \left(1 - u_{i,j}^{4}\right) \qquad \forall i,j,t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \tag{4.93}$$

$$V_{i,0,a_{i,j}^k} = V^{Initial} \qquad \forall i, a_{i,j}^k \in (\mathbf{I},\mathbf{T}) \tag{4.94}$$

Constraints to ensure that only a single $u_{i,j}^l$ equals one

$$\sum_{l=1}^{4} u_{i,j}^l = 1 \qquad \forall i, j \in (\mathbf{I},\mathbf{J}), l = 1,2,3,4 \tag{4.95}$$

Constraints to ensure that only a single $v_{i,j}^l$ equals one

$$\sum_{l=1}^{4} v_{i,j}^l = 1 \qquad \forall i, j \in (\mathbf{I},\mathbf{J}), l = 1,2,3,4 \tag{4.96}$$

The computed time points

$$t^A = \frac{S^{\max} - S_{i,j,a^k}^{Rem}}{r_i} + a_{i,j}^k \qquad \forall i, j, t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \tag{4.97}$$

$$t^{A'} = \frac{S^{\min} - S_{i,j,d^L}}{f_i} + d_{i,j}^L \qquad \forall i, j, t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \tag{4.98}$$

$$t^C = \frac{V^{\max} - V_{i,j-1,a^k}}{h_i} + a_{i,j}^k \qquad \forall i, j, t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \tag{4.99}$$

$$t^{C'} = \frac{V^{\min} - V_{i,j,t^C}}{g_i} + t_{i,j}^{mdc} \qquad i, j, t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \tag{4.100}$$

$$t_{i,j}^{mdc} = \mathbf{max}\left\{ t_{i,j}^C, D_i^{V^{\max}} + a_{i,j}^k \right\} \qquad \forall i, j, t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \tag{4.101}$$

The time required for motivation of worker i to reach the upper bound starting from an initial level $V^{Initial}$

$$D_i^{V^{\max}} = \frac{V^{\max} - V^{Initial}}{h_i} \qquad \forall i \in (\mathbf{I}) \tag{4.102}$$

Minimum time interval between the two assignments to the same workstation

$$a_{i,j}^k - d_{i,j}^L \geq \Omega_{i,j} \qquad \forall i, j, t \in (\mathbf{I},\mathbf{J},\mathbf{T}) \tag{4.103}$$

Types of variables

$$x_{i,j,t}, v_{i,j}^l, u_{i,j}^l \in \{0,1\} \quad \forall i, j, t \in (\mathbf{I},\mathbf{J},\mathbf{T}), l = 1,2,3,4 \tag{4.104}$$

$$t_{i,j}^A, t_{i,j}^{A'}, a_{i,j}^k, t_{i,j}^C, t_{i,j}^{C'}, d_{i,j}^L, t_{i,j}^{mdc}, D_i^{V^{UB}} \geq 0 \quad \textbf{Integer} \quad \forall i, j \tag{4.105}$$

$$\mathrm{S_{i,j,t}}, \mathrm{V_{i,j,t}} \in \{\mathrm{R}\}(\text{real number}) \tag{4.106}$$

4.6.4 Numerical example

The same data used in the previous example for the nonlinear model can be used to verify the above formulation. The computational results for the four scenarios are summarized in Table 4-2.

Table 4-2. Computational results (linear model)

	Total Lost Production Time				
	α				
	0.5	*0.6*	*0.7*	*0.8*	*0.9*
Case 1	104.5	86.1	67.6	49.2	30.7
Case 2	89.5	76.5	63.5	50.5	37.5
Case 3	88.4	75.5	62.7	49.9	37.0
Case 4	65.9	72.1	78.2	84.4	95.5

Similar to what observed in the numerical example for the nonlinear model, the results in this table (Table 4-2) show that for an equal weight factor of 0.5 the total lost production time (LPT) decreases from case 1 (do nothing) to case 4 (rotate workers every 8 time units after time 32). As the weight factor of skill improvement increases, the implementation of job rotation may even increase the total LPT i.e., reduction in productivity. Figure 4-9 compares the total lost production times associated with the four cases described in the above numerical example.

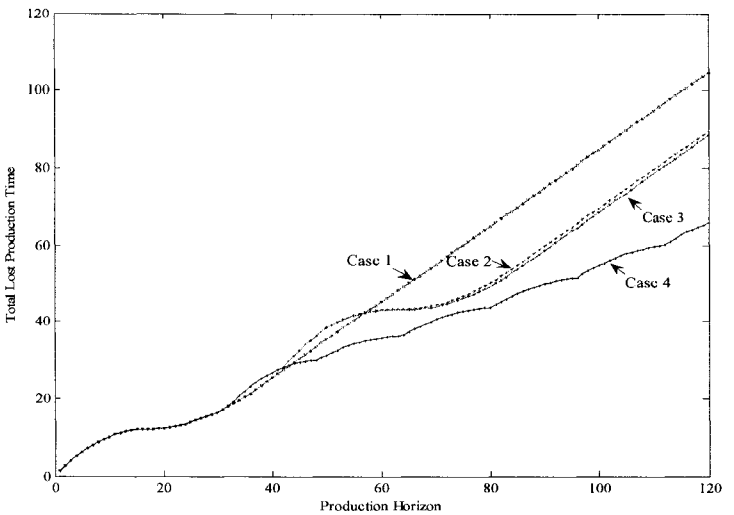
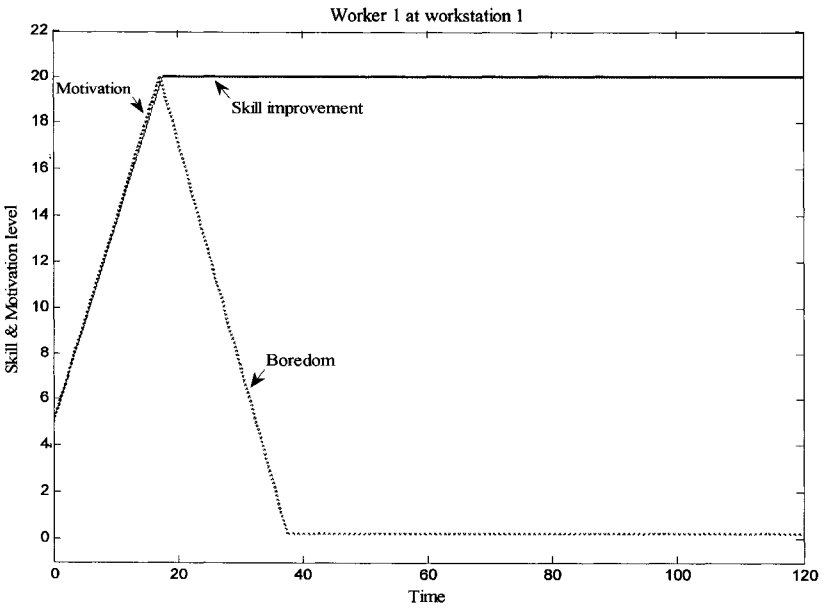
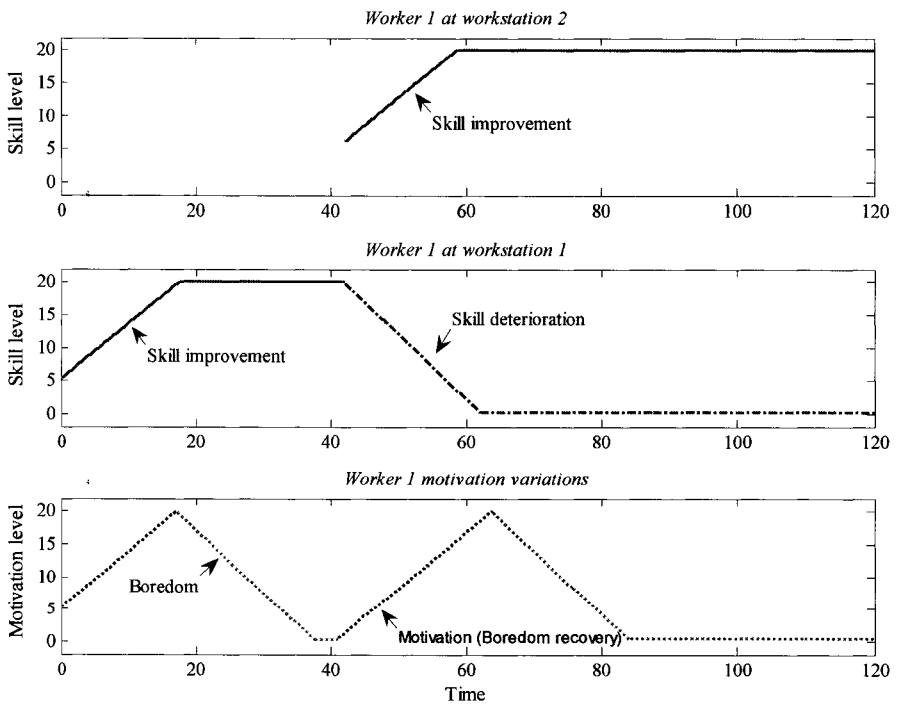


Figure 4-9. Comparison of the total lost production times over the production horizon (linear model)

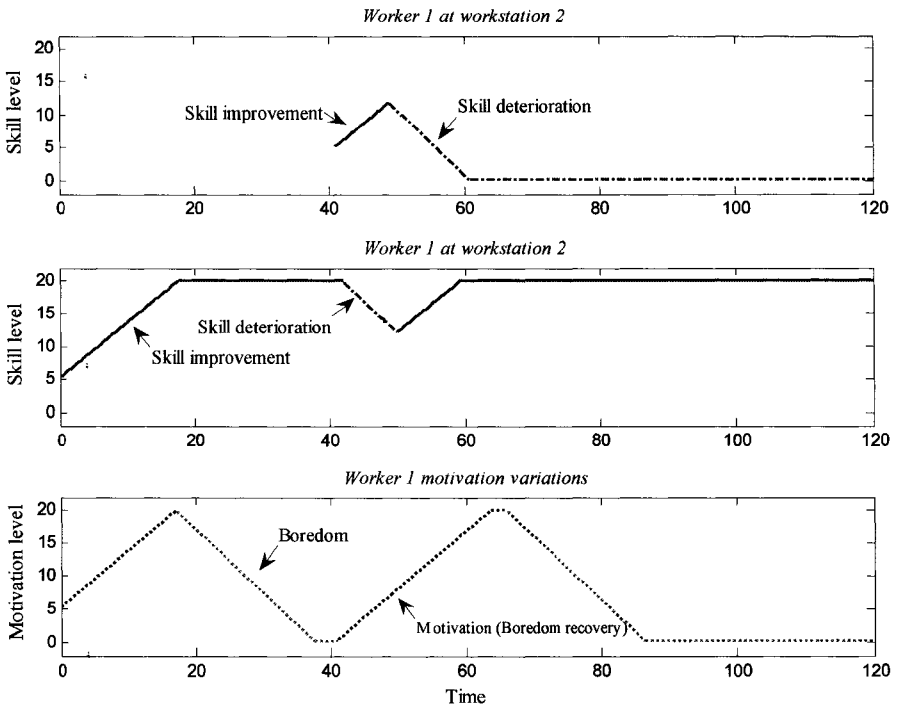
The variations of worker 1 motivation and skill in the four cases are presented in Figure 4-10. It should be pointed out that the job rotation model developed in this chapter requires: a) input data, and b) an efficient solution method for large scale problems encountered in an industrial setting. However, the input data may not be always available and they have to be reasonably generated. The next section is dedicated to dealing with this issue.



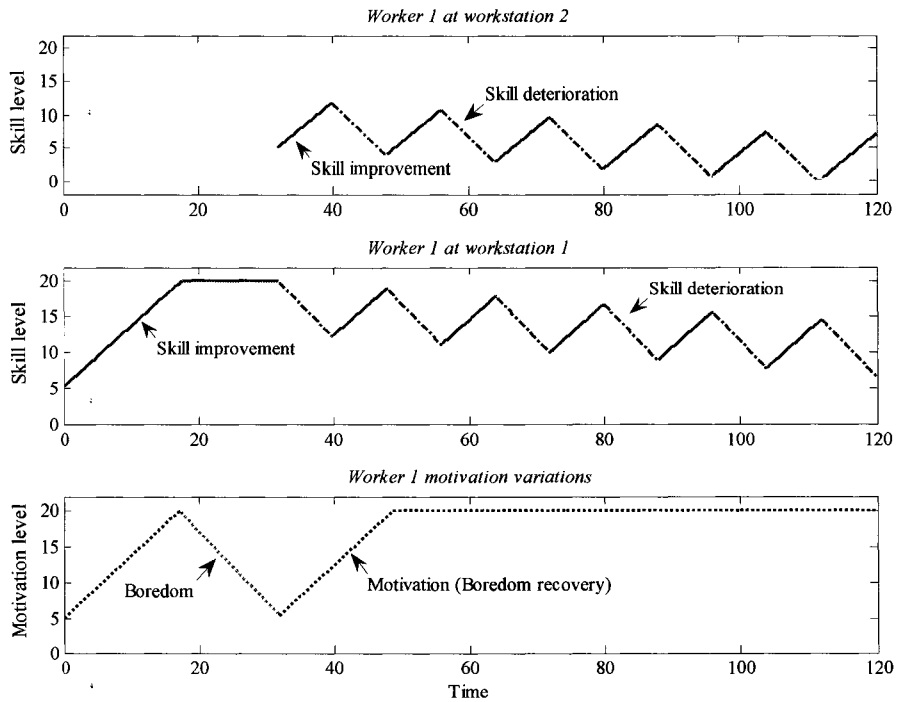
(a) Case 1



(b) Case 2



(c) Case 3



(d) Case 4

Figure 4-10. Graphical illustration of the numerical example (linear model)

4.7 A desiccation support system to implement job rotation strategies

The job rotation models proposed in this chapter requires input data including slopes of learning, forgetting, motivation, and boredom. As stated in section 4.3.2, workers' learning and forgetting coefficients could be measured using historical data (Brown and Heathcote, 2003; William *et al.*, 2008). The slopes of workers motivation and boredom could also be estimated using historical data. However, if such data is not available, the use of a Decision Support System (DSS) that integrates a group of Bayesian models and a series of mathematical formulations to respectively predict workers boredom and skill variations is suggested. Such a system will help managers to make informed decisions with regard to job rotation strategies.

This chapter presents a DSS that includes an objective function, a skill measurement component, a number of Bayesian models, and a database that stores worker's personal and other necessary information. Framework of the proposed DSS is presented in Figure 4-11.

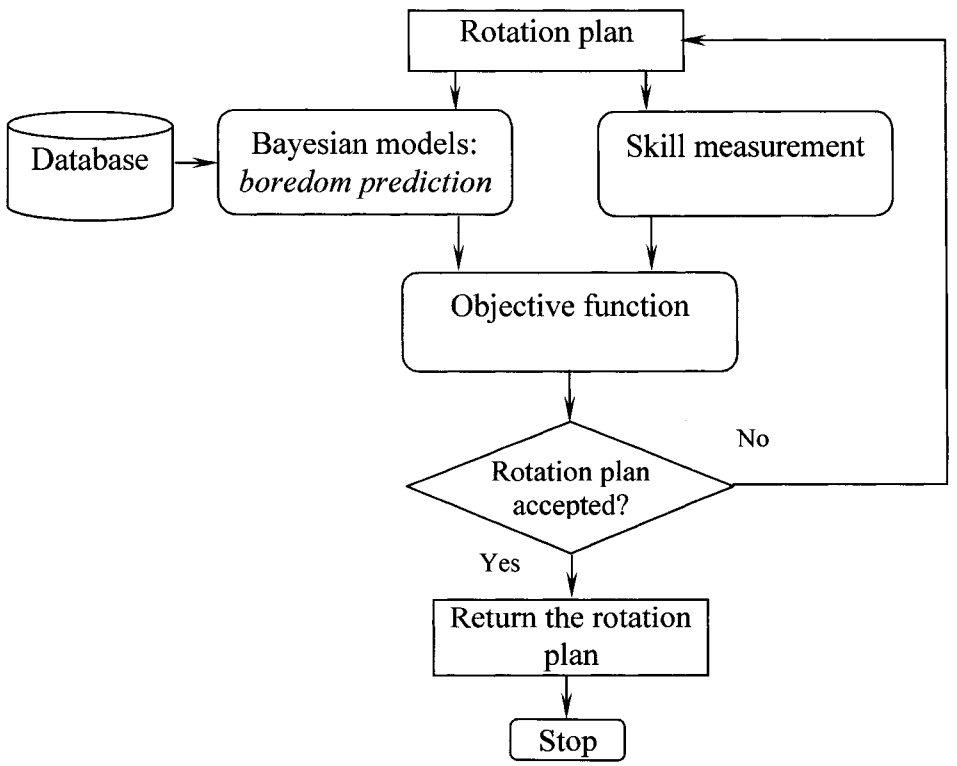


Figure 4-11. Framework of a decision support system to implement job rotation strategies

The objective function utilized in the DSS is the same as that presented in equation 4.15. This function simultaneously considers workers' skill and motivation variations to calculate the total lost production time during a production horizon. The skill measurement module computes the workers skill variations over a production horizon using related equations presented in sections 4.3.2 and 4.4.2. This module attempts to depict the variations in worker's skill in response to a particular rotation plan.

The Bayesian model module contains a number of probabilistic models each predicting the boredom level of a particular worker during a production horizon. Parts of the input data for the models including the length(s) and location(s) of the assignment(s) are extracted from the current rotation plan. The rest of the (required) data (e.g., workers personal information) are collected from a database. The Bayesian model for job boredom presented in Chapter 3 is suitable for measuring and monitoring human boredom. This model is slightly modified in order to be used as a part of the described DSS. The adapted Bayesian model for job boredom is described as follow.

In the job rotation model proposed in Chapter 4, it is assumed that as a worker continues to perform the same task over time, his/her boredom level is increased. Therefore, the increment in worker's motivation/boredom is a function of time he/she spends in a particular workstation performing a repetitive task. To address this assumption, a new variable, *Tenure*, is introduced in the Bayesian model for job boredom. This variable is defined as the term or period during which a worker is assigned to a particular workstation. It takes three states, Long, Medium, and Short. Furthermore, it is also essential to account for the temporal aspects of some other variables such as worker's capacity which may affect work's boredom especially in long-term assignments. In view of the above, the topology of the new (dynamic Bayesian) model for boredom is presented in Figure 4-12.

4.7.1 Information flow in the DSS

In the proposed DSS, first a rotation plan that satisfies workers skill variation constraints is introduced. The rotation plan is then simultaneously sent to the Bayesian models module (to predict the workers boredom variations) and to the skill measurement module (to calculate the variations of workers skill). The required information

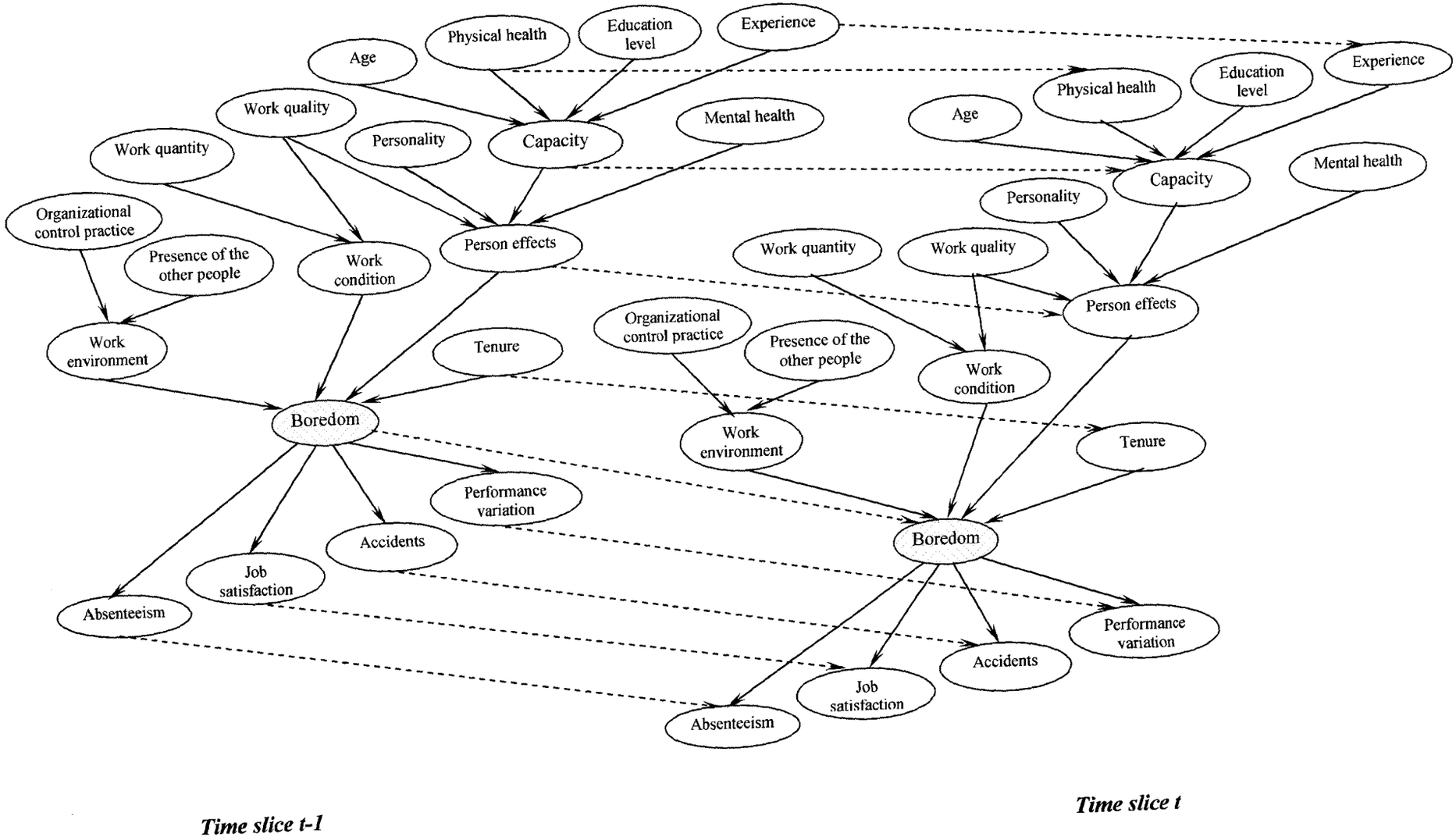


Figure 4-12. Modified dynamic Bayesian model for boredom at work

pertaining to each worker including the location of the assignment(s), the length of the tenure(s), etc is extracted from the proposed plan and input to the (workers) Bayesian models. The other essential information including workers' personal information (e.g., education, age, and experience), work condition(s), and work environment(s) are transferred from a database to the Bayesian models module. Once all the available information is collected, the inference process for each Bayesian model is carried out to predict works motivation (boredom) variations during a pre-decided horizon time. As soon as the predicted workers' motivation data become available, they will be use as the input to the objective function module.

On the other hand, the skill measurement module computes the skill of workers at each time slice during pre-decided horizon time according to the proposed rotation plan.

The outputs of the two modules, Bayesian models and skill measurement, are then sent to the objective function component to evaluate the total lost production time associated with the current plan. If the corresponding value of the lost production time is acceptable, the process is ended and the DSS returns the rotation plan. Otherwise, another job rotation is generated and evaluated as described above.

Chapter 5

Solution to the Job Rotation Problem: A New Metaheuristic Approach for Combinatorial Optimization and Scheduling Problems

In Chapter 4, two versions of the proposed job rotation model, linear and nonlinear, were presented. Because of the complexity of the two models, large problems cannot be easily solved using commercial software. For this reason, the application of two conventional heuristics, simulated annealing and genetic algorithm, to the nonlinear version of the proposed model is investigated. The preliminary results on several randomly generated job rotation problems indicate that the two conventional methods perform relatively poor both in terms of solution quality and computational times. To alleviate this problem, in this chapter, a generic framework of a new metaheuristic is presented to solve combinatorial optimization problems. Then, based on the proposed framework, a search algorithm is tailored to solve the job rotation problems. The general framework of the new metaheuristic will be presented in section 5.1 followed by description of the search algorithm for job rotation model in section 5.2. The performance of the proposed metaheuristic and comparisons with the two conventional methods, simulated annealing and genetic algorithm, are presented in section 5.4.

The generic framework of the metaheuristic is a hybrid approach containing a simulated annealing algorithm with memory and evolution-based diversification to solve combinatorial optimization problems. The proposed hybrid algorithm combines different features of several well known heuristics. The integrated components of the algorithm include a simulated annealing module, one long-term and two short-term memories, and a genetic algorithm component, and a blockage removal feature. The simulated annealing

module is the driver of the algorithm that utilizes the two short-term memories to simultaneously intensify the search in the area surrounding good solutions and to avoid cycling. The first memory is a tabu list and the second memory is a seed memory list that keeps track of good solutions visited during the last iterations. Under certain conditions, the best solutions in the seed memory lists are added to a population list to setup a long term memory. Once the population is entirely set, individuals are combined via an evolutionary operator to generate a new population from which an offspring might be selected as an initial solution for the subsequent iteration. The blockage removal feature is used to resolve possible deadlock situations that may occur during the search process.

Following the detailed description of the proposed method, the application of the proposed method the job rotation problem is presented in this chapter.

5.1 The proposed hybrid approach: SAMED

This study attempts to take advantage of several essential ingredients of different search heuristics, particularly the Simulated Annealing (SA), Tabu Search (TS) and Genetic Algorithm (GA), and integrated them into a metaheuristic algorithm. The distinctive features of the proposed method are the use of both negative (inhibitory) and positive (reinforcement) short term memories and a new evolution-based diversification mechanism. The algorithm containing the above components is named SAMED, standing for SA with Memory and Evolution-based Diversification.

5.1.1 The rationale

The conventional SA approaches attempt to avoid being trapped in a local optimum by occasional moves to solutions with higher cost. However, as the search progresses and the transition probability declines, the possibility that the search gets trapped in local optima may increase. To alleviate this problem, several methods such as the multi-start approaches (Aarts *et al.*, 1994) and adaptive temperature control mechanism (Kolonko, 1999; Azizi and Zolfaghari, 2004) have been suggested in the literature.

Another drawback of the SA approach is the lack of memory. Some researchers have used a combination of TS and SA to overcome this deficiency (Osman 1993, Zolfaghari and Liang 1999). The main advantage of adding a tabu list to a SA algorithm is that the

search would have a memory that provides an inhibitory (negative) feedback to the search. Nevertheless, such memory may not be sufficient as it does not have any positive feedback from the search history (El-Bouri *et al.*, 2007).

The plain GA in its original form, called standard GA hereafter, suffers from several shortcomings. In fact, the success of a GA method in solving a problem relies primarily on the efficiency of the problem-specific operators. As a result, numerous efforts have been made in the past to design more efficient operators for GA. Premature convergence is another drawback of the standard GAs (Wang and Zheng, 2001). For this reason, many GA methods are unable to converge to an optimal solution.

The above deficiencies present in the original GA give rise to hybrid GAs. Their most common form is a two-level search technique where a local search is used to upgrade the entire or part of the newly generated offspring. The members of the population are then operated on in the next generation by an evolutionary recombination operator. Thanks to the complementary properties of GAs and local search heuristics, hybrid approaches often outperform either method operating alone. However, they still require excessive computational effort. Furthermore, in such algorithms deciding when to terminate the local search steps is often a difficult task.

In view of the above, this study proposes a new metaheuristic approach, SAMED, a hybrid SA algorithm with memory that uses an evolution-based diversification. The SAMED includes five main components: a SA module, three types of memories, and a GA-based crossover component. The first two types of memory are of short-term and are implemented by a *tabu list* and a *seed memory list*. The third memory is of long-term and its function is fulfilled by a *population list*. Unlike other hybrid algorithms, the core component of the proposed method is a simulated annealing module that continuously searches the solution space. The SA module merely stops functioning to allow the search be diversified by the GA component in due course.

5.1.2 Description of the proposed method

SAMED starts with an initial solution that could be generated randomly or by means of other heuristics. The search procedure conducted by the algorithm includes two separate phases, intensification and diversification. During intensification phase, SAMED utilizes

two types of short-term memory to explore the neighbourhood of potential good solutions. Once a solution is accepted, part of the solution that has been modified to generate a candidate solution is preserved temporarily in a tabu list to avoid cycling as well as to reduce the size of the neighbourhood under investigation. The other short term memory which is called seed memory list keeps track of the solutions accepted in the current iteration. Once the short-term memory reaches a predetermined size, a *short iteration* is said completed. At this stage, if the current solution is not the best among those stored in the seed memory list, SAMED redirects the search toward previously visited good solutions by selecting the best solution in the seed memory list. In other words, the iteration's best solution is treated as the starting point for the subsequent short iteration. The same solution is also stored in a long-term memory which is called *population list*. In the end of each short iteration and before starting a new iteration both short-term memories are emptied. The above steps are repeated until the long-term memory becomes full i.e., the population list is entirely setup. The completion of the long term memory marks the end of the *long iteration*. In the end of long iterations, SAMED evaluate the performance of the search by comparing the quality of the overall best solution with that in the end of the previous iteration. In the case of improvement, the search continues by utilizing the last iteration's best solution as the initial solution for the next iteration. Otherwise, the elite solutions preserved in the population list are recombined via an evolutionary operator to generate a new population (i.e., beginning of the diversification phase).

The quality of the best solution among the newly generated offspring solutions is compared with the overall best solution. If it is superior to the current overall best solution, the offspring solution is used as a new initial solution for the subsequent iteration. Otherwise, a random number (between zero and one) is compared with the current value of (normalized) Complement of Temperature (CT) function. If the random number is less than the current value of the CT function, the new population is scanned and the first offspring whose cost is different from that of the iteration's best solution is selected as the initial solution for the next iteration. Afterward, the temperature is reset to its initial value.

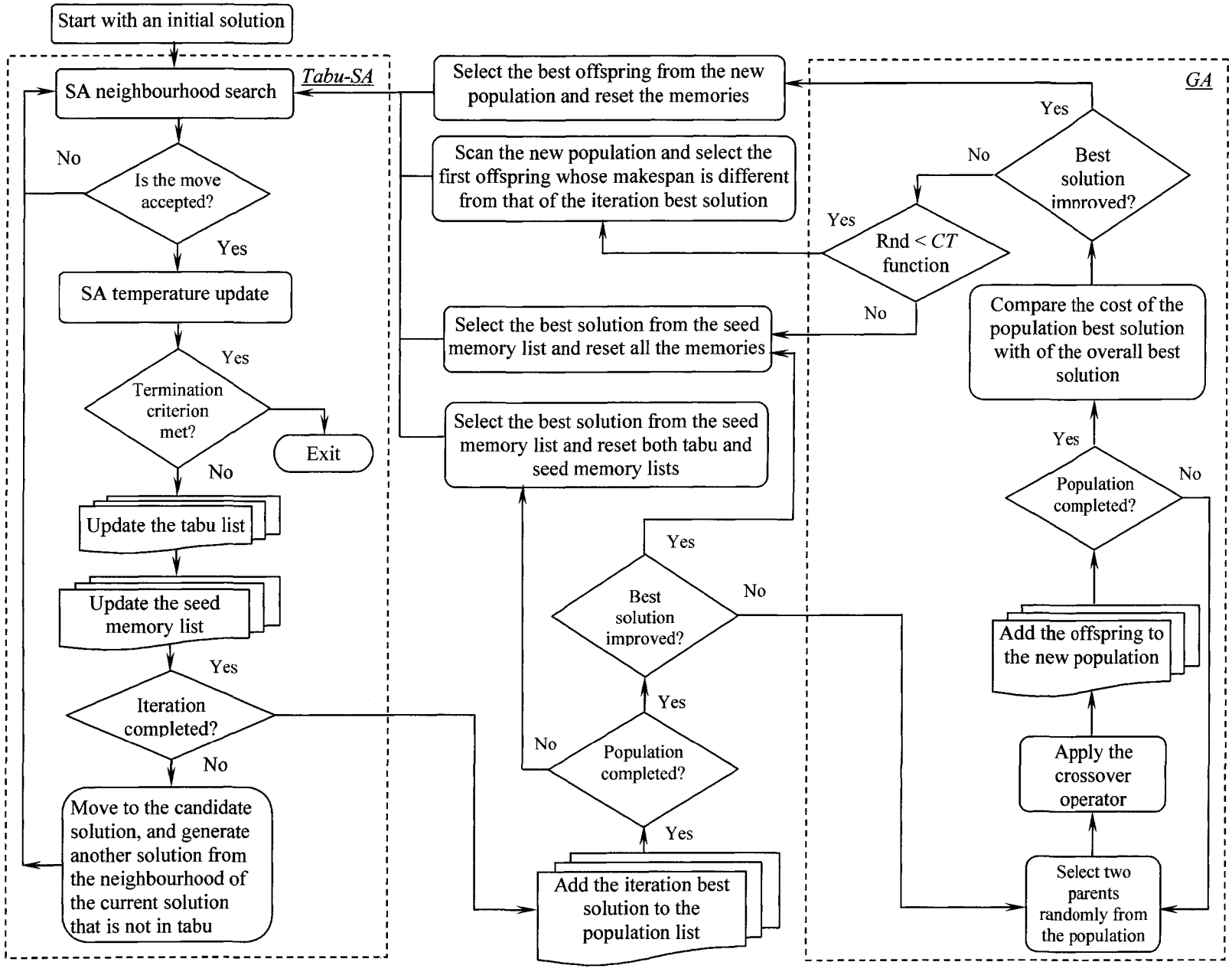


Figure 5-1. Framework of the SAMED

If the random number is greater than the value of CT function, the search continues with the current iteration's best solution without updating the temperature.

The compliment of temperature function (CT function) discussed above could be presented as

$$CT\ function = 1 - f(\theta_i)$$

where $f(\theta_i)$ is the normalized temperature function, and θ_i represents the temperature in iteration i.

The framework of the SAMED is presented in Figure 5-1.

5.2 Application of the SAMED to the job rotation problem: SAMED-JR algorithm

To solve the proposed job rotation model, a search algorithm is developed based on the generic framework of the SAMED (Figure 5-1). The developed algorithm is called SAMED-JR from now on, with JR indicating job rotation. The SAMED algorithm is composed of several components: a Simulated Annealing (SA) module, three types of memory, a Genetic Algorithm (GA) component, and a blockage removal feature. To apply the SAMED-JR to the job rotation problem these components need to be customized. The SAMED-JR components are described as follows.

5.2.1 The SA module

The SA module is the core component of the SAMED-JR algorithm. It carries out the essential tasks such as neighbouring solution generation, solution evaluation, and solution acceptance or rejection decisions. The SA module includes several key ingredients including solution representation, neighbourhood function, cooling schedule, and cost function. These elements are customized for the job rotation problem and presented in the following sections.

5.2.1.1 Solution representation

In the SA module, a solution to the job rotation problem is represented by a string composed of several segments. Each segment of the string represents a time slice and it

includes several elements called "genes". Each element of a segment represent a worker operating in a workstation at a given time slice. Scanning a segment from left to right each element of the segment is represented by an integer number that corresponds to a worker's identification number. Furthermore, the location of each element (i.e., worker) on the segment represents the assigned workstation for each worker. Figure 5-2 illustrates a solution to the job rotation problem with five workers, five workstations, and for a production horizon of length five.

Scanning the chromosome from left, the first position on the first segment from left begins with number 1 indicating that worker "1" has been assigned to workstation "A" at time t_1.

For simplicity and to reduce the computational burden, the chromosome representing solutions to the job rotation problem is converted to a matrix. In this more compact representation format, solutions are represented by an $m \times n$ matrix in which m and n represent the number of workstations and time slices in production horizon respectively. Entities of the matrix characterize workers performing operations during the production horizon (Figure 5-3).

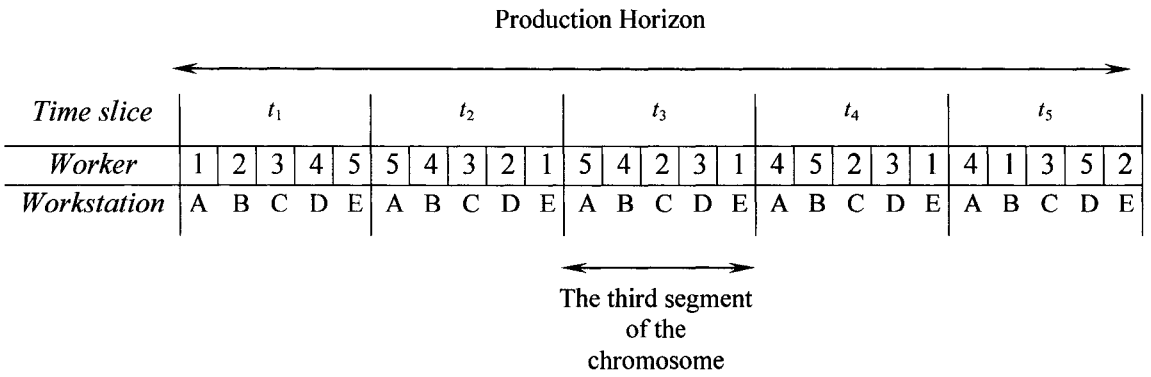


Figure 5-2. An illustrative example of a solution to the job rotation problem

More precisely, entities of each row in the matrix are the collection of corresponding elements taken from all segments. For instance, the first row is the collection of the first members of each segment in the chromosome, the second row is the collection of the second members,..., and the last row is the collection of the last elements of each segment.

Solution to the job rotation problem

The chromosome representation of the job rotation problem and its relationship with matrix representation is illustrated in Figure 5-3.

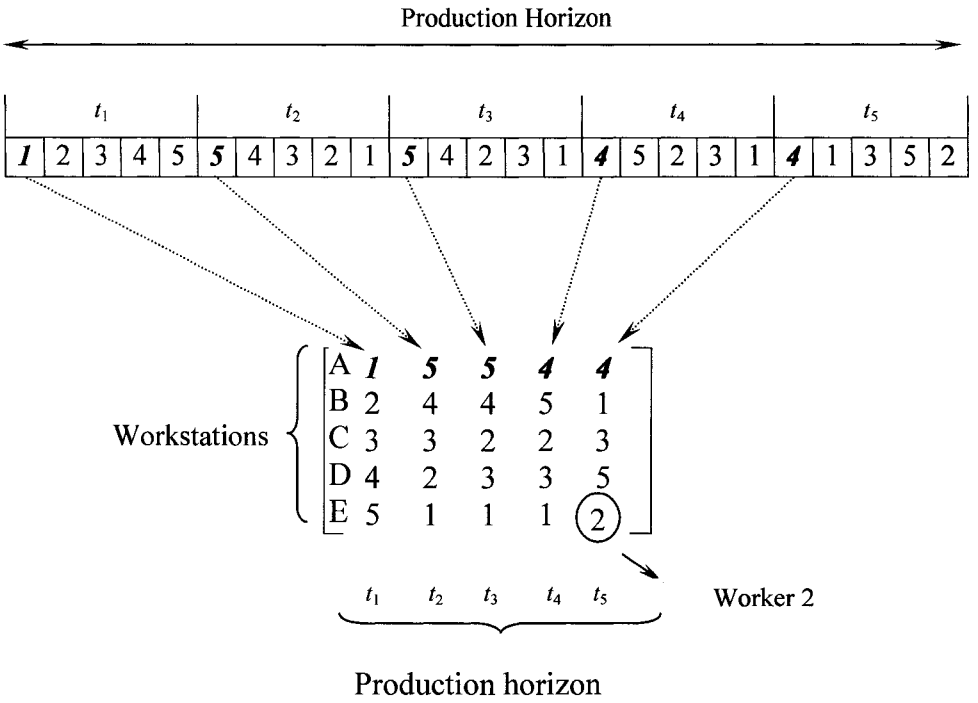


Figure 5-3. Representing a solution to job rotation by matrix

5.2.1.2 Neighbourhood structure

A neighbouring solution is generated by exchanging the workstations of two employees at a particular time slice. To generate a neighbouring solution, first a time slice is selected randomly (i.e., randomly selecting a column). Then in the corresponding column of the

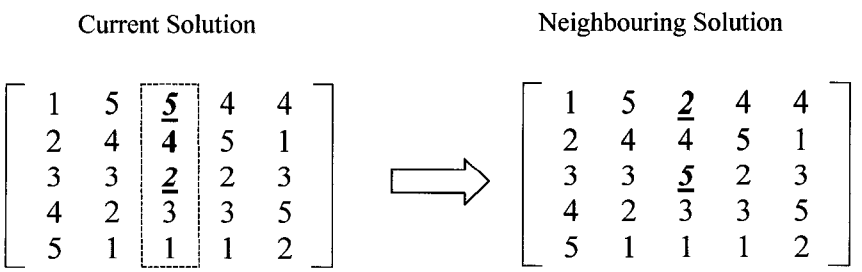


Figure 5-4. Neighbourhood generation mechanism

current solution, positions of two randomly selected entities are swapped. If the swap generates an infeasible solution, another solution is generated and evaluated. For instance, in Figure 5-4 at time slice three (t_3) (i.e., column 3 the position of two workers, worker 5 and 2, are swapped to generate a neighbouring solution.

5.2.1.3 Cooling schedule

The following simple logarithmic function is used as the cooling schedule.

$$\theta_{l+1} = \frac{\theta_1}{1 + \lambda \ln(L)} \tag{5.1}$$

where θ_1 is the initial temperature, L represents the move number, and λ is a control parameter.

5.2.1.4 Cost function

The cost function is the total lost production time over the production horizon.

5.2.2 The GA component

To generate a new population, two parents (e.g., parent1 and parent2) are selected randomly from the population list one at a time. Then, the *single point* crossover operator is used to generate members of the new population (Reeves, 1995). To apply the crossover operator, a column and a row are selected randomly and the two matrices (i.e., parent1 and parent2) are cut as shown in Figure 5-5. Then, the first segment of each parent matrix is taken and the second segment is replaced by that of the other parent.

The illustrated cut in Figure 5-5 passes through more than one column (column 2 and column 3). The rationale behind cutting the matrix in such a way is to make sure that all parts of the solution space will have a chance to be visited during the search process. For instance, if only one column is selected to cut the matrix, the configuration of all genes in the associated time segment (see Figure 5-3) will not have a chance to alter during the search process. Consequently, in such a case a large portion of the solution space may not be investigated.

Solution to the job rotation problem

Given the solution representation of job rotation problem, there is a good chance that combining the two parents yield an unfeasible offspring. In such a case, a repairing algorithm is required to convert an unfeasible offspring to a feasible one.

Violation of the following two constraints during the reproduction phase are the root causes of generating infeasible solutions. The first constraint prevents assigning a worker to more than one workstation at each time slice. The second constraint prevents reallocation of the worker to the same workstation for a certain amount of time. If the first constraint is violated during the crossover operation, the repairing algorithm replaces the entity (ies) in the selected column (below the selected row) that causes the infeasibility of the solution with new elements to make the solution legitimate.

If the solution could not be converted to a feasible solution by replacing the entities underneath of the selected row then, another pair of parents is selected and operated to produce the offspring matrices.

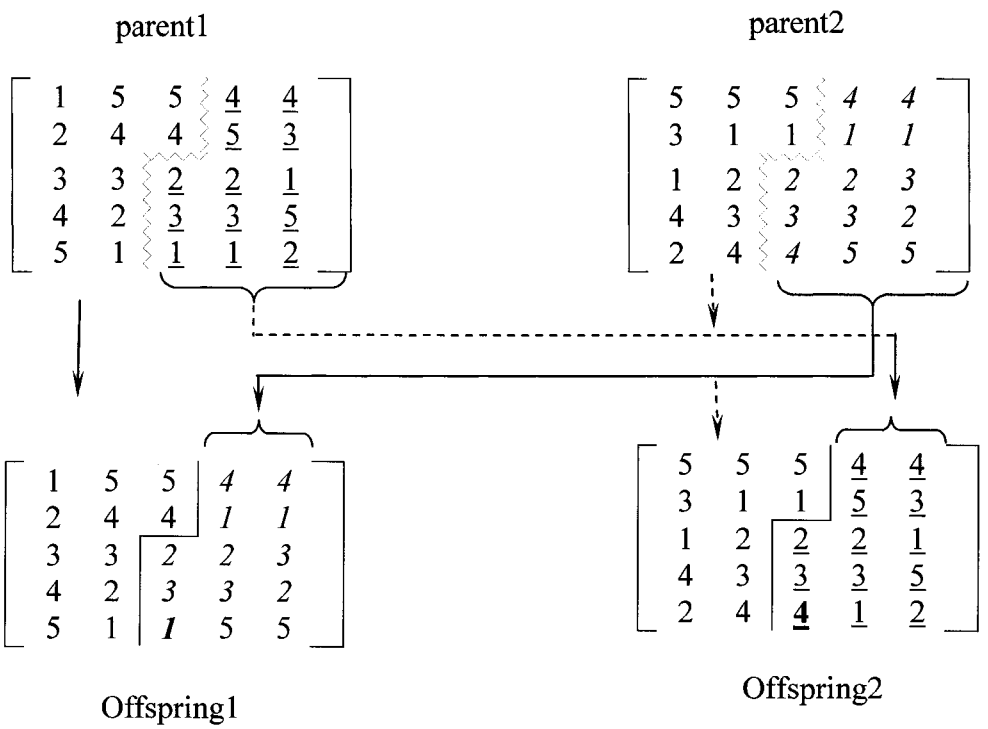


Figure 5-5. Crossover operation

5.2.3 The blockage removal

The purpose of this feature in the SAMED-JR is to resolve possible deadlock situations that may occur during the search procedure. In such undesired situation due to the limited number of un-forbidden (i.e., not in tabu) solutions, the SA module cannot find an acceptable solution in the neighbourhood of a current solution. To resolve the described deadlock situation, the SAMED-JR algorithm after a number of unsuccessful attempts, Q, will accept the last examined feasible solution regardless of its cost. Here the upper bound for the number of (unsuccessful) attempts is assumed to be 1000 (i.e., Q=1000).

5.3 The search procedure

The SAMED-JR begins the search using a *feasible* initial solution. To generate a neighbouring solution, a column (i.e., time slice) on the matrix of the current solution is selected randomly. Then on the selected column, the location of the two randomly selected entities (i.e., two workers) is exchanged. If the candidate solution is unfeasible, then another solution is generated and examined.

Given the cost of the candidate solution and the current state of the temperature, the SA component of the SAMED-JR decides whether to accept or reject the solution. If the solution is rejected, another solution is generated and evaluated. In the case of acceptance, first the positions of the two entities that have been modified to generate the candidate solution is added to a *tabu matrix* and then the entire solution is added to a seed memory. Except for the first iteration, each time when a solution is accepted, the cost of the candidate solution is compared to that in seed memory list. If the preserved solution in the memory is inferior to the candidate solution, it is replaced by the candidate solution. Otherwise, the move to the candidate solution is performed without updating the seed memory. The above steps are repeated until the tabu matrix reaches a predetermined size. At this point a *short iteration* is said to be completed.

At the end of short iterations, the SAMED-JR examines the performance of the search by comparing the quality of the short iteration's best solution with that in the end of the previous *long iteration* where a long iteration is defined as a combination of several predetermined short iterations. If the quality of the previous long iteration has improved, the search continues by using the short iteration best solution as the initial

solution for the next iteration and both tabu matrix and seed memory are emptied. Otherwise, the short iteration's best iteration is added to a long-term memory which is also called *population* list. Then, both memories are emptied and the search continues by selecting the iteration's best solution as the initial solution for the next iteration.

If the situation persists (i.e., there is no improvement with respect to the previous long iteration) for the entire long iteration, the SAMED-JR continues building up the population. In such a case, at the end of a long iteration at which the population is entirely set up, the SAMED-JR uses the crossover operator of the GA component to generate a new population.

Upon completion of the new population the quality of the best member of the population is compared with that of the overall best solution. If it improves the quality of the overall best solution, it is selected as the initial solution for the next iteration. Otherwise, a random number between zero and one is generated and compared with the current value of the Complement of Temperature function (CT function). If the random number is less than the value of CT function, the new population is scanned and the first individual whose cost is different from that of the iteration best solution is selected as the initial solution for the next iteration and the temperature is set to the initial value (i.e., high level). Otherwise, the search continues with the iteration best solution without changing the temperature. At the end of long iterations, all the memories are emptied regardless which solution has been selected as the starting point.

The structure of the SAMED-JR is illustrated in Figure 5-6.

Solution to the job rotation problem

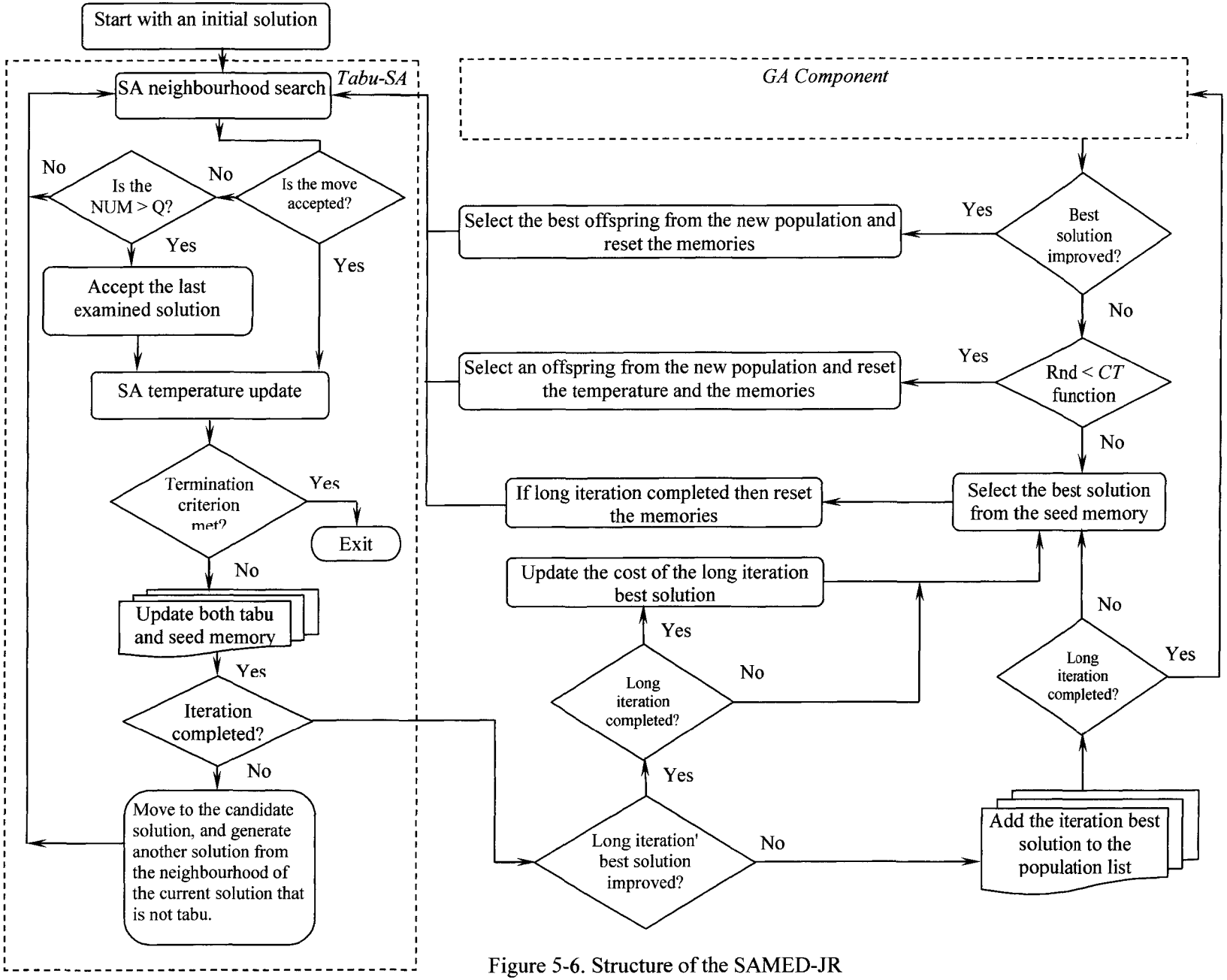


Figure 5-6. Structure of the SAMED-JR

5.4 Conventional simulated annealing and genetic algorithm

To evaluate the performance of the SAMED-JR on the proposed job rotation problem, two popular heuristics, Simulated Annealing and Genetic Algorithm, have been also developed.

The same elements utilized in SA component of the SAMED-JR (i.e., solution representation, neighbourhood structure, and cooling schedule) have been used to develop a stand-alone SA.

As in the GA component of the SAMED-JR algorithm, the developed stand-alone genetic algorithm utilizes the single point crossover operator described in section 5.2.2. In the GA component of the SAMED-JR during reproduction phase each time a pair of parents are selected *randomly* to be operated by the crossover. However, the stand-alone GA uses the *Roulette Wheel* (Man *et al.*, 1999) selection mechanism to generate a new population of solutions. The roulette wheel technique selects two individuals with a probability biased toward the chromosomes with better fitness values. Each member of the population is assigned a sector of a roulette wheel that is proportional to its fitness value (Figure 5-7). Then the wheel is spun to select a parent.

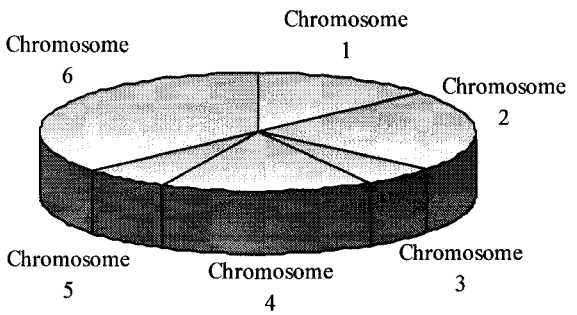


Figure 5-7: Chromosome fitness proportion in roulette wheel

5.5 Computational results

To assess the performance of the proposed method, SAMED-JR, on the job rotation problem, 16 benchmark problems have been generated randomly. The problems are generated by randomly selecting the parameters such as workers learning, forgetting,

motivation, and boredom slopes. The size of the problems varies from five workers/workstations and 50 units of time (the length of the production horizon) to 20 workers and 120 units of time (see Appendix B for details).

To solve the problems, the three algorithms, i.e., SAMED-JR, SA, and GA have been coded on Visual Basic. The programs were run on a Pentium 4 PC with 3.3 GHz CPU. For fair comparisons, identical parameters are used in the stand-alone version of SA and the SA component in the SAMED-JR. For instance, in both the stand-alone SA and the SA component of the SAMED-JR, the initial temperature and the control parameter are set to 10 and 2, respectively. In the stand-alone GA, the population size, the probability of crossover, and the probability of mutation are respectively set to 60, 0.9, and 0.2; the initial population is generated randomly for each test problem. Both the stand-alone SA and SAMED-JR have been run from randomly generated initial solutions,

The parameters of the SAMED-JR are as follow. The short-term memory for the test problems JR-1 to JR-12 is five; for JR-13 to JR-16 it is set to 10. The long-term memory for the test problems JR-1 to JR-4 is 10; for JR-5 to JR-16 it is 30. The computational time (second) is limited to 180 for JR-1 and JR-2; 240 FOR JR-3; 300 for JR-4 to JR-10, JR-13, and JR-14; 420 for JR-11, JR-12, and JR-15; and 600 for JR-16. It is worthwhile to mention that the parameters of the SAMED-JR were simply set at what seemed to be sensible values after some fairly small-scale experiments.

The computational results are summarized in Table 5-1. In this table, the numbers represent the total lost production times (LPT) and the time at which the solution was found (in parenthesis). The results in this table clearly indicate that the SAMED-JR outperforms the other two methods by finding solutions with lower LPTs. For the four benchmark problems with different sizes, the comparisons of the convergence behaviour of the three methods are illustrated in Figure 5-9. This figure shows that the SAMED-JR is much faster than the two conventional algorithms. A typical solution to the JR-1 benchmark problem with a cost (LPT) of 62.035 time units is also presented in Figure 5-8 where rows and columns represent workstations and time slices, respectively. Each entity of the matrix corresponds to a worker operating in a workstation at a certain time slice.

Table 5-1. Computational results

Problem		Method		
Size ($n \times T$)	Name	SAMED-JR	SA	GA
5×50	JR-1	62.70 (170)	82.64(152)	81.62(178)
5×70	JR-2	86.45(173)	113.0(144)	116.88(1)
5×100	JR-3	122.41(170)	158.34(101)	157.22(234)
5×120	JR-4	147.01(224)	189.83(283)	194.2(3)
10×50	JR-5	151.0(292)	182.68(218)	180.47(284)
10×70	JR-6	209.4(293)	252.13(11)	252.43(299)
10×100	JR-7	301.93(300)	358.0(74)	359.4(70)
10×120	JR-8	366.04(298)	427.98(49)	429.3(127)
15×50	JR-9	246.03(300)	275.98(117)	272.93(299)
15×70	JR-10	343.66(297)	384.42(138)	385.97(288)
15×100	JR-11	491.36(417)	548.67(375)	549.75(408)
15×120	JR-12	591.3(416)	657.57(359)	658.73(49)
20×50	JR-13	341.85(289)	368.5(170)	369.55(295)
20×70	JR-14	485.2(296)	514.58(264)	516.98(91)
20×100	JR-15	690.02(399)	736.81(34)	740.55(15)
20×120	JR-16	828.27(576)	887.2(446)	889.69(51)

Note: n = number of workers/workstations, T = number of time units in production horizon

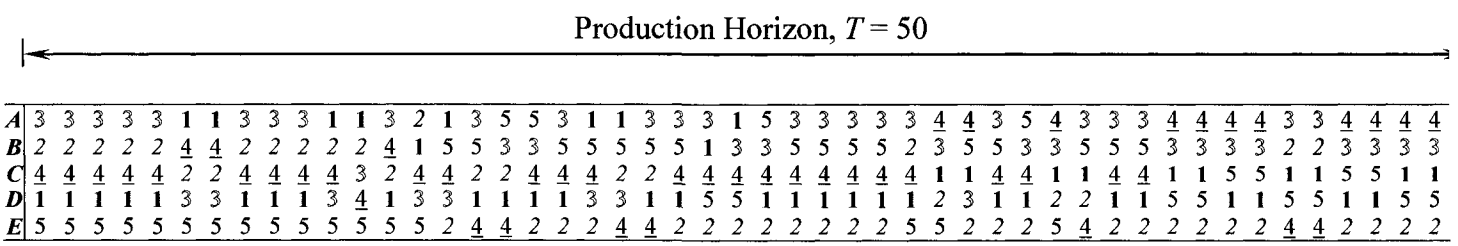
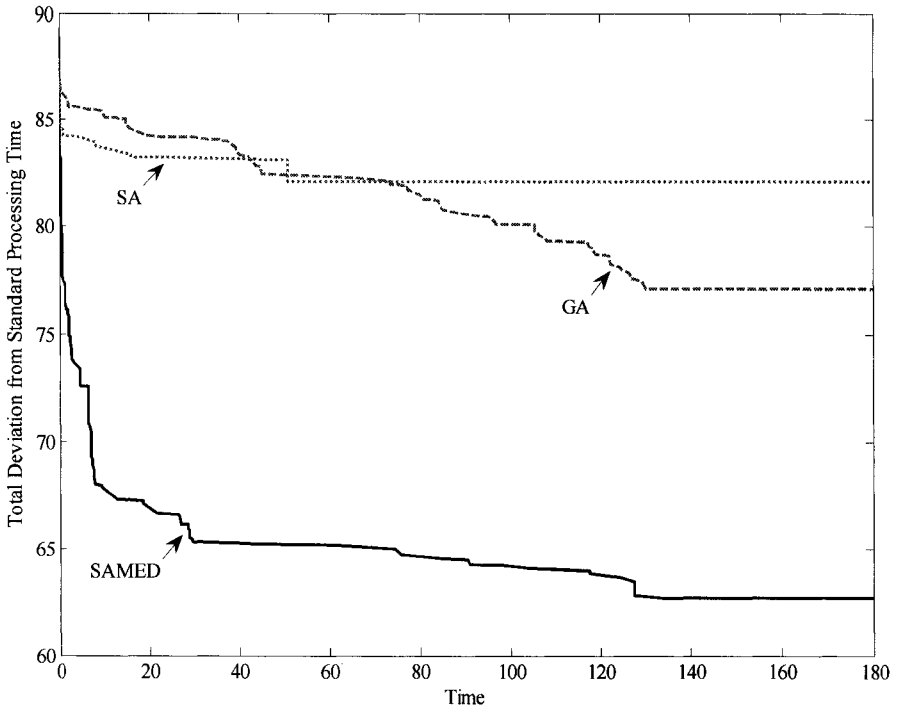
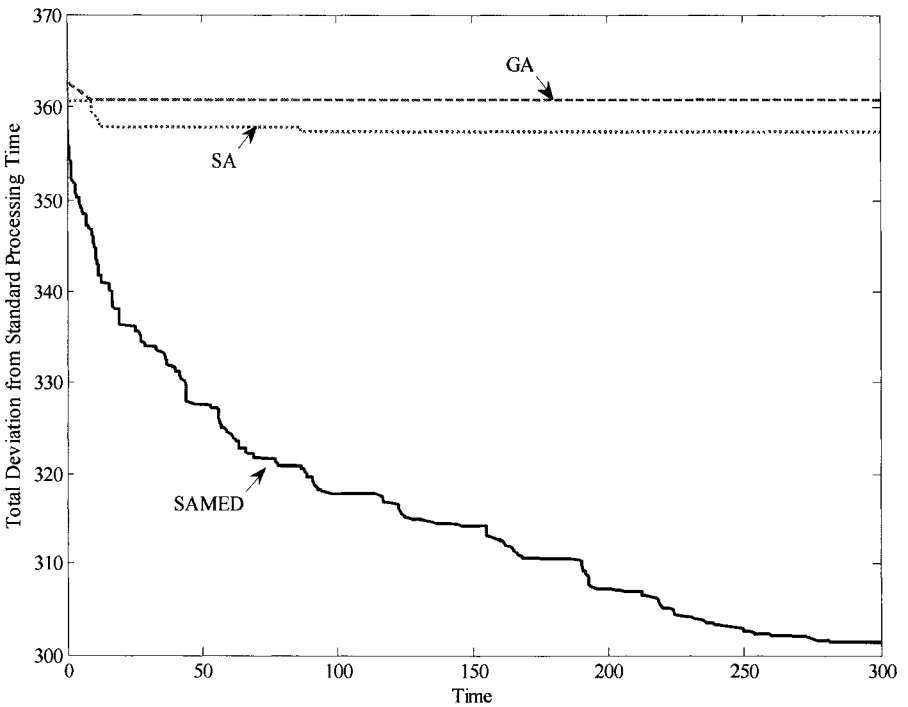


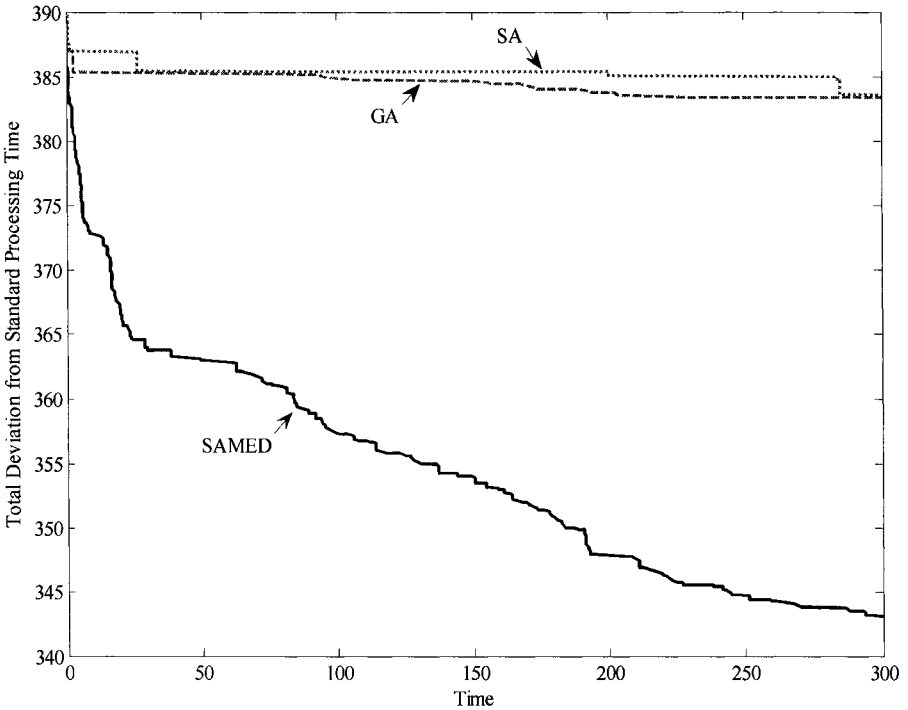
Figure 5-8. An example solution to the problem JR-1



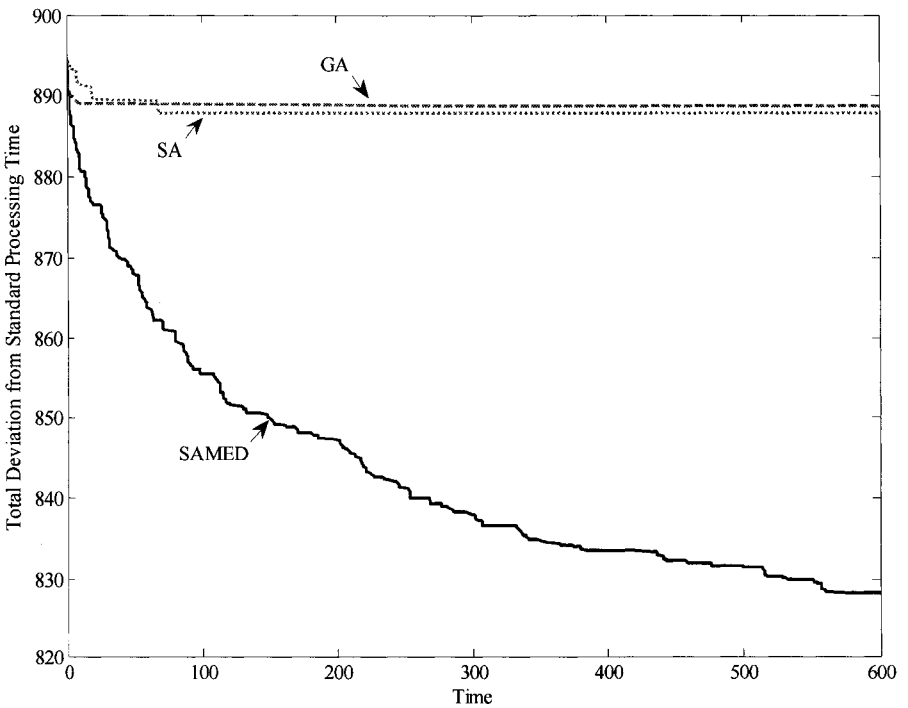
(a) Test problem JR-1



(b) Test problem JR-7



(c) Test problem JR-10



(d) Test problem JR-16

Figure 5-9.Convergence comparison of the SAMED-JR with conventional simulated annealing and genetic algorithm

Chapter 6

Machine Utilization: job shop and flow shop scheduling problems

As stated in sections 1.1 and 2.7, the goal of this thesis is to improve manufacturing productivity by addressing issues related to both human and machine resources. The human issue (such as boredom) can be partially addressed by solving the job rotation model presented in chapter 5 and the machine issue can, to a certain extent, be dealt with by solving job scheduling problems to be presented in this chapter.

Scheduling decisions largely affect machine utilization hence, they are essential for proper machine utilization. Scheduling, in general, could be defined as decision making process to arrange activities in order to meet specific objectives, to optimize resource allocations, and to execute tasks given set of constraints. Various scheduling problems that occur in manufacturing industries have been investigated in the literature. They are inherently complex and often categorized as combinatorial Non-polynomial (NP) hard problems. These problems are very difficult to solve using existing heuristics or conventional techniques.

Scheduling problems in two popular manufacturing systems, job shop and flow shop, are investigated in this thesis. To solve the problems, several hybrid search algorithms based on the proposed generic framework (chapter 5), SAMED, will be tailored and presented in this chapter.

A job shop production system provides high level of product variety at the expense of high cost and long lead-times while a flow-shop system provides low-cost and shorter lead times at the expense of product variety. In a classical job shop environment there is not a clear flow of the parts within the production facility. Each part (or job) that is

produced in the facility may require a different set of operations and/or a different sequence of operations. Usually, jobs are processed in one machine at a time and machines process one job at a time. In a flow shop system, products follow a well-defined sequence of operations through a series of dedicated machines.

6.1 Application of the SAMED to the job shop scheduling problem

To examine the performance of the proposed generic framework (SAMED) on the job shop scheduling, an NP-hard problem, a search algorithm (SAMED-JSS) has been developed. The problem statement and application details are described below.

6.1.1 Job shop scheduling problem

The job shop scheduling problem studied in this research is described as follows. Given a set of jobs and a set of machines, each job consists of a sequence of operations that has to be executed in a specific order. Each operation has to be performed on a given machine without interruption. The objective is to find the schedule to minimize makespan, subject to the following constraints: each machine can handle at most one job at a time and the operation precedence should not be violated for each job. The job shop scheduling problem can be formulated as follows:

Let

$E = \{1, 2, 3, \ldots, n^e\}$ denote the set of jobs;

$M = \{1, 2, 3, \ldots, m^l\}$ denote the set of machines

$O_{e,l}$ = operation of job e on machine l

$P_{e,l}$ = processing time of job e on machine l

$x_{e,l}$ = start time of job e on machine l

Then the job shop scheduling problem can be modeled as

$$\textit{Minimize} \quad C_{\max} = \max_{e,l}\left(x_{e,l} + p_{e,l}\right) \tag{6.1}$$

Subject to

$$x_{e,l} \geq 0 \qquad \forall f, e \in E; l, r \in M \tag{6.2}$$

$$x_{e,r} \geq x_{e,l} + p_{e,l} \qquad \text{If } O_{e,l} \text{ precedes } O_{e,r} \tag{6.3}$$

$$x_{f,l} \geq x_{e,l} + p_{e,l} \qquad \text{If } O_{e,l} \text{ precedes } O_{f,l} \tag{6.4}$$

$$x_{e,l} \geq x_{f,l} + p_{f,l} \qquad \text{If } O_{f,l} \text{ precedes } O_{e,l} \qquad (6.5)$$

This problem has been considered as one of the most stubborn combinatorial problems. Though "exact" solutions may be obtained for small problems using branch and bound methods, it is unrealistic to find "exact" solutions for medium and large problems facing decision makers in real applications. Some heuristic methods such as Shifting Bottleneck Procedure (SBP) have thus been developed. However, as Balas *et al.* (1995) pointed out, SBP suffers from difficulty in the re-optimization phase and infeasible solutions.

6.1.2 Specification of the SAMED-JSS components

To apply the proposed algorithm to the job shop scheduling problem, the following components need to be "customized".

6.1.2.1 Encoding scheme

Encoding a schedule is the key to the success of a metaheuristic method in solving any scheduling problems. In the context of job shop scheduling, several encoding schemes such as operation based, job based, machine based, disjunctive graph based, random keys, binary and permutation have been proposed in the literature. Some of them such as binary and permutation may yield an infeasible solution if the solution (generated by these schemes) is subject to a minor or major perturbation. As a result, a repairing algorithm is usually required to transform an infeasible solution into a feasible one.

In this study, an operation-based representation proposed by Bierwirth (1996) is adopted. This approach uses an un-partitioned permutation of m-repetitions of job numbers to represent a solution as a string. Each job is a set of operations that has to be processed on m machines. In this form of representation, each job number occurs m times in the permutation. Scanning the permutation from left to right, the k^{th} occurrence of a job number refers to the k^{th} operation in the technological sequence of this job. This will ensure that an operation is not scheduled unless all its technological predecessors have been scheduled. For instance, the following string represents a solution for a problem with four jobs and four machines:

4 2 4 1 1 3 2 4 1 4 1 3 3 2 2 3

Each job consists of four operations and is repeated four times. The fifth element of the string is 1. This number refers to the second operation of job 1 because it appears for the second time. Similarly, the ninth and the eleventh elements of the string respectively refer to the third and the fourth operations of job 1.

6.1.2.2 SA component

In practice, three basic ingredients are needed to apply a simulated annealing algorithm: a cooling schedule, a cost function, and a neighbourhood structure. In this study, the simulated annealing component embedded in the framework of the proposed heuristic uses the following cooling schedule proposed by Van Laarhoven *et al.* (1992).

$$\theta_{i+1} = \frac{\theta_i}{1 + \frac{\theta_i \ln(1+\delta)}{3\sigma_i}} \tag{6.6}$$

where θ_i is the temperature in iteration i, σ_i is the standard deviation of the previously visited solutions and δ is an empirical distance parameter.

The cost function is the makespan of the job shop scheduling problem and a neighbouring solution is generated by swapping the positions of two non-identical jobs on a string. Given the definition of neighbouring solution, the size of the neighbourhood can be calculated by the following equation:

$$\text{Neighbourhood size} = m^2\left(\sum_{k=1}^{n-1}(n-k)\right) \tag{6.7}$$

where m and n denote the number of machines and jobs, respectively. Verification for this equation can be found in Appendix C.

6.1.2.3 The GA module

In the proposed heuristic, the GA module contains only a random selection mechanism and a crossover operator. Once the long term memory is completed a new population is generated by consecutively applying the crossover operator to the randomly selected parents from the current population. For the operation-based encoding scheme used in

this study, the precedence preservative crossover (PPX) has proven to be efficient (Bierwirth, 1996). The advantage of this method is that the newly generated individual could be directly decoded as a schedule without any modification (i.e., always generate feasible solutions). Following this method, a template vector h of length $m \times n$ (m and n denote the numbers of machines and jobs, respectively) is filled with random elements from set {1, 2}. This vector is then used to define the order in which elements are drawn from parent 1 and parent 2. The selected element from one parent is appended to the offspring string and then the corresponding element is deleted from both parents. This procedure repeats until both parent strings are emptied and the offspring contains all the involved elements.

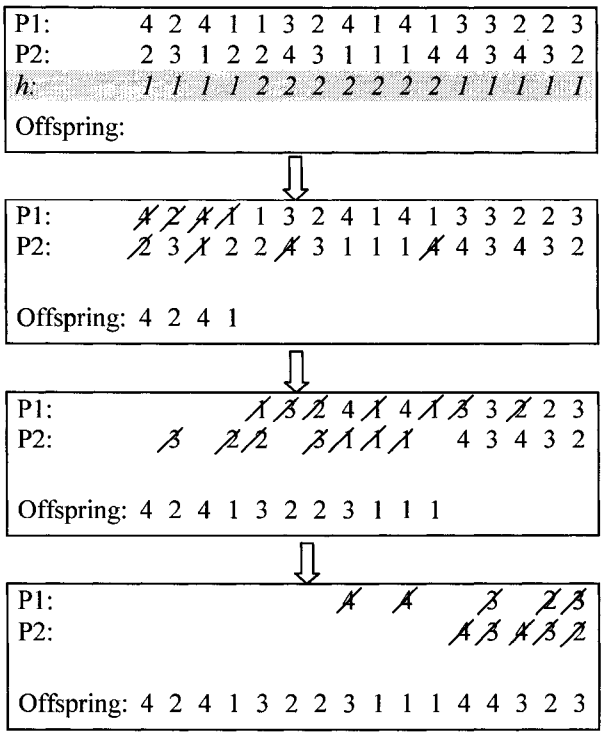


Figure 6-1. An illustrative example of the crossover process

Figure 6-1 shows an illustrative example with four jobs and four machines. P1 and P2 are respectively parent 1 and parent 2 that have been selected randomly. In this example, the first four elements of template vector h are 1. Therefore, the first four elements from parent 1 (P1) are appended in offspring string and corresponding elements in parent 2 which are respectively sixth, first, eleventh and third elements are deleted.

The second set of the random numbers in vector *h* (from the fifth to the twelfth elements) is 2. Consequently, the first seven elements from the remaining parts in parent 2 (elements 3 2 2 3 1 1 1) are added to the offspring string and then corresponding elements in parents 1 and 2 are deleted. The above procedure is repeated until the offspring string contains all the involved elements.

6.1.2.4 The search procedure

The algorithmic steps of the SAMED-JSS is as follow

Step 1. Initialization

1. Generate an initial solution
2. Substitute the current solution with the initial solution
3. Calculate the makespan of the current solution
4. Set:
    a) Short-term memory size
    b) Long-term memory size
    c) Computational time

Step 2. Solution generation and evaluation

Generate a candidate solution by swapping the position of two non-identical jobs (selected randomly) on the chromosome of the current solution.

IF either of the two jobs is in the tabu list then, return to the beginning of Step 2;

ELSE evaluate the cost of the candidate solution.

IF the candidate solution is rejected, return to the beginning of Step 2;

ELSE update the overall best solution and the temperature.

IF the stop criterion is met, stop; otherwise go to next Step.

Step3. Memory update

(1) Add the two randomly selected jobs into the tabu list and the entire candidate solution into the seed memory list

(2) Move to the candidate solution

IF the tabu list is not full, return to Step 2;

ELSE add the best solution from the seed memory list into the population list.

IF the population is full, go to Step 4;

ELSE replace the current solution with the best solution in the seed memory list, clear all the information stored in both tabu and seed memory lists, and then return to Step 2.

Step 4. Diversification

(1) Compare the overall best solution with that in the end of the previous long iteration

IF the overall best solution has improved, go to (4) of this Step;

ELSE generate a new population using those solutions stored in the population list and the crossover operator in the GA component.

IF the best member of the new population improves the quality of the overall best solution, substitute the current solution with the best member of the new population and go to (5) of this Step;

ELSE generate a random number (between zero and one).

(2) Compare the random number with the current value of the Complement of the Temperature function (CT function)

IF the random number is greater than or equal to the current value of the CT function, go to (4) of this Step;

ELSE scan the new population and select the first offspring whose makespan is different from the makespan of the iteration's best solution.

(3) Substitute the current solution with the selected offspring and set the temperature in the SA component at initial value and go to (5) of this Step.

(4) Replace the current solution with the best solution in the seed memory list

(5) Clear all the information stored in the memories and return to Step 1

Solutions in the new population generally inherit elements from the iteration's best solutions visited previously. Starting the search from one of these solutions not only guides the search away from local optima, it also increases the chance of improving the best solution in a short period of time. However, because of the structural similarity between these solutions and the best solution found so far, there is also a chance that the search will end up with the same best solution. To reduce the chance of revisiting the same (best) solution, temperature is set to high level at the beginning of the new iteration.

Increasing the temperature may encourage the search to extend the area of the exploration (diversification). Moreover, raising the temperature immediately lowers the value of the CT function (Complement of Temperature function). A low CT function value will prevent the reallocation of the initial solution prematurely (intensification). This could be regarded as the second effect of the temperature change at the beginning of the new iteration.

According to the cooling schedule utilized in this study, the temperature declines from the high of 0.95 with respect to the standard deviation of the previously visited solutions (Van Laarhoven *et al.*, 1992).

For this cooling schedule, the CT function could be defined as:

$$\text{CT function} = \Omega \times (1 - \theta_i) \tag{6.8}$$

where θ_i is the temperature in iteration i and Ω is called the diversification coefficient. Based on the above equation, setting the temperature at a high level any time during the search will automatically retune the value of the CT function to its lowest level. By the same token, as the search continues and the temperature declines, the magnitude of the CT function increases. A higher value of CT function enhances the possibility of success in situations that a decision has to be made on changing the search direction by utilizing a new initial solution.

In equation (6-8), the diversification and/or intensification phase of the algorithm is affected by Ω. The diversification level can be reduced by using a Ω value ranging between 0 and 1 since this will reduce the possibility of re-initialization (intensification). Obviously, assigning Ω with a value greater than one would raise the chance of selecting a new initial solution from the population (diversification).

With the above components, the temporal complexity of the SAMED-JSS algorithm is

$$O((mn)^3(\ln n)(L_{ms} \times S_{ms})) + P^{GA}O(L_{ms}mn) \text{ per iteration,}$$

where m and n represent the number of machines and number of jobs respectively, L_{ms} is the size of the long-term memory, S_{ms} is the size of the short-term memory, and P^{GA} is the probability that GA component is called.

In the above complexity expression, the second term is clearly of lower complexity than the first term, therefore the overall complexity of the SAMED-JSS could be presented as:

$$O((mn)^3(\ln n)(L_{ms}S_{ms})) \tag{6.9}$$

Detailed analysis of the SAMED-JSS temporal complexity can be found in Appendix D.

6.1.3 SAMED-LB: the SAMED with local search and blockage removal

The proposed algorithm in section 6.1.2, SAMED-JR, is a very general method that can be easily tailored and applied to other combinatorial optimization problems. Before computationally examining the above algorithm, an extended version of it, with a local search feature and a blockage removal capability, is presented. The extended version is especially tailored for job shop scheduling.

The local search module has been developed based on the job shop scheduling neighbourhood structure proposed by Nowicki and Smutnicki (1996) and embedded to the structure of the SAMED-JSS algorithm. On certain occasions, this module investigates the possibility of swapping the position of specific jobs on the critical block of a current solution. This may lead to a fast improvement in the quality of the solution. The operation of the local search module is separated from that of the SA component of the SAMED-JSS as depicted in figure 6-4.

The purpose of adding this feature is to investigate the effect of a problem-specific local search (i.e., a local search that uses some problem-specific information) on the proposed method.

The blockage removal feature is applied to resolve the deadlock situations that may occur during the search procedure. The components related to the two new features are described in the following.

6.1.3.1 Local neighbourhood search module

Given the Gantt chart representation of a solution, the local search begins its operation by identifying the *critical block chain* in a given solution. A critical block chain is composed of several blocks each containing one or more adjacent operations on a machine. In each solution, one or more block chains could be identified. The longest is called the critical block chain. In the example shown in Figure 6-2, the shaded three jobs on four machines form the critical block chain.

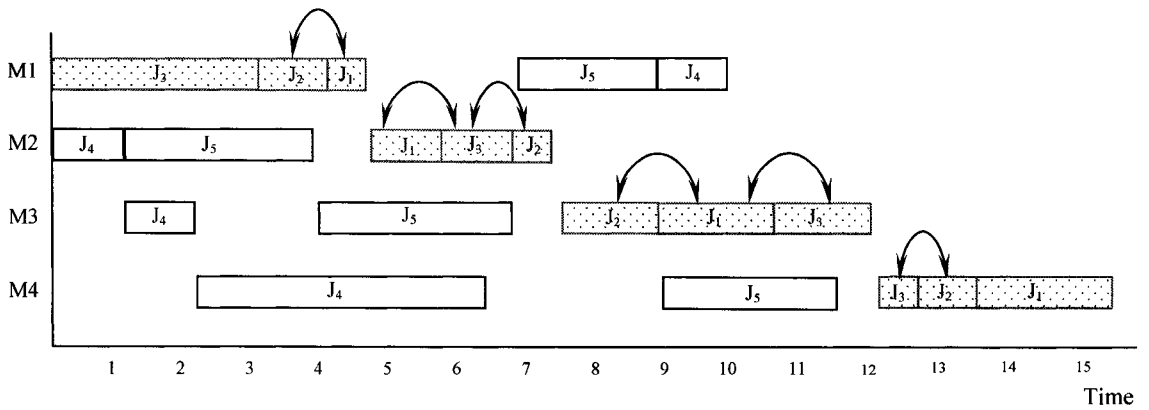


Figure 6-2. An example of a critical block chain and neighbourhood moves

Once the critical blocks are determined, the following neighbourhood moves are generated according to the position of each block on the critical block chain. Assuming that there are more than two adjacent operations on each block, the last two operations on the first block and the first two operations on the last block are swapped to generate two solutions. For the rest of the blocks, the decision as to which two operations to swap is made by generating a random number between zero and one. If the random number is less than 0.5, the last two operations on the block are swapped; otherwise, the first two are selected to swap. If a block contains only two operations, the two operations are swapped regardless of their positions on the critical block. Finally, in the case that a block contains only one operation, no swap is necessary.

Once all the possible neighbouring solutions according to the above procedure are generated, the best solution with lowest makespan is determined. Selecting the best neighbouring solution marks the end of the current iteration of the local search module.

An example problem with five jobs and four machines is illustrated in Figure 6-2. In this example, the size of the possible neighbourhood moves is 4. The first solution is generated by exchanging the positions of the last two jobs on the first block (jobs 1 and 2 on machine M1). The second and the third solutions could be generated by either swapping the positions of the first or the last two jobs on blocks 2 and 3. The last solution is generated by swapping the first two jobs on the fourth block (jobs 3 and 2 on M4).

The pseudo code of the local neighbourhood search module is shown in Figure 6-3.

```
Neighborhood_ search (iteration best solution)

  Determine the critical block chain
  Determine the number of blocks on the critical block chain (TotalNomOfBlocks[a])
  Set:
          TmpBest solution[b]= iteration best solution
      NomOfBlocks[c]=1
  Do while NomOfBlocks < TotalNomOfBlocks+1
  {
      If first block then
          New Solution= swap last two operations of block incurrent solution
          If makespan(New Solution) < makespan(TmpBest solution) then
              TmpBest solution= New Solution
          End if
      Else
          If last block then
              New Solution= swap first two operations of block in current solution
              If makespan(New Solution) < makespan(TmpBest solution) then
                  TmpBest solution= New Solution
              End if
          Else
              Generate a random number (Rnd)
              If Rnd < 0.5 then

                  New Solution= swap last two operations of block in current solution
                  If makespan(New Solution) < makespan(TmpBest solution) then
                      TmpBest solution= New Solution
                  End if
              Else
                  New Solution= swap first two operations of block in current solution
                  If makespan(New Solution) < makespan(TmpBest solution) then
                      TmpBest solution= New Solution
                  End if
              End if
          End if
      End if

      NomOfBlocks= NomOfBlocks + 1
  }
  Loop

  Return TmpBest solution
```

[a]: Total Number of Blocks, [b]: Temporary Best Solution, [c]: Number of Blocks

Figure 6-3. The pseudo code of the local search module

6.1.3.2 Blockage removal component

During the search procedure, the chance of being trapped in a deadlock is quite high when the temperature is low and the number of jobs in the tabu list is increasing. In a deadlock situation, the Tabu-SA component may not be able to find any acceptable solutions in the neighbourhood of a current solution. As a remedy for such undesirable conditions, a blockage removal feature is necessary.

With this feature, whenever the number of trials conducted by the SA component to find an acceptable solution exceeds the value calculated by the following equation, the last feasible solution is accepted regardless of the cost.

$$\text{Max. number of unacceptable moves} = \left[\frac{m \times n \times (n-1)}{2}\right] \times \gamma \tag{6.10}$$

where m and n denote the number of machines and jobs, respectively and γ is a coefficient that controls the magnitude of the maximum number of unacceptable moves.

The justification for equation (6.10) is as follows. In a simple pair-wise exchange neighbourhood generation scheme, $n\times(n\text{-}1)/2$ solutions can be generated by swapping the positions of two jobs on a machine. Given the number of machines in the problem of interest, a total of $m\times n\times(n\text{-}1)/2$ neighbouring solutions can be generated. This value, which is much less than the size of the neighbourhood in this study (see equation (6.7)), is selected as the upper bound of the number of consecutive unacceptable moves.

In equation (6.10), the value of the parameter γ affects the number of allowable trials to find an acceptable solution in the neighbourhood of a solution. While assigning a value between zero and one to γ may lead to a quick exploration for such solution, the parameter γ with an assigned value greater than one allows more search in the neighbourhood under investigation.

6.1.3.3 The search procedure

Integrating the above two components with the other elements of SAMED-JSS, SAMED-LB is developed and described as follows. The steps of the algorithm conducted by both Tabu-SA and GA components are the same as the one described in the earlier version.

Therefore, to avoid repetition, only the aspects related to the new components in the search procedure will be discussed.

At each stage during the search when the length of the tabu list reaches its predetermined size (i.e., short iteration completed), the overall best solution is compared with that at the end of the previous (long) iteration (Figure 6-4). If the quality of the solution has improved, the local search will be applied to the best solution of the current iteration. Then, the seed memory list will be updated. If the (long) iteration is completed, both the overall best (Min-makespan) and the iteration best solutions will be updated. The search continues by selecting the best solution from the seed memory list as an initial solution for the subsequent iteration. If the iteration is not completed, the best solution among those stored in the seed memory list will be selected as the initial solution for the next iteration without updating the overall best and the iteration's best solutions.

At the end of the short iteration, even if the overall best solution remains unchanged, the neighbourhood search module may still contribute to the search procedure. In such cases, the iteration best solution is compared to that at the end of the previous iteration. If the quality of the solution has improved, the local search will be applied to the iteration best solution. Subsequently, the seed memory list is updated and the makespan of the overall best solution is compared to that at the end of the previous iteration. In the case of improvement, if the iteration is completed, both the overall best solution (Min-makespan) and the iteration best solution are replaced with their new values. The search continues by selecting the best solution in the seed memory list as an initial solution for the subsequent iteration. If the long iteration is not completed then, without updating the overall and iteration best solution the search continues by selecting the solution in the seed memory list.

If the quality of the overall best solution remains unchanged, the iteration best solution (that may have been improved by the local search) is added to the population list. Then, the status of the population is verified. If the population is completed, the GA module is activated to generate a new population. Otherwise, the search continues by selecting the best solution from the memory list as an initial solution for Tabu-SA component. Finally, if no improvement is observed either in the overall best or the iteration's best solution, the last (short) iteration's best solution is added to the population

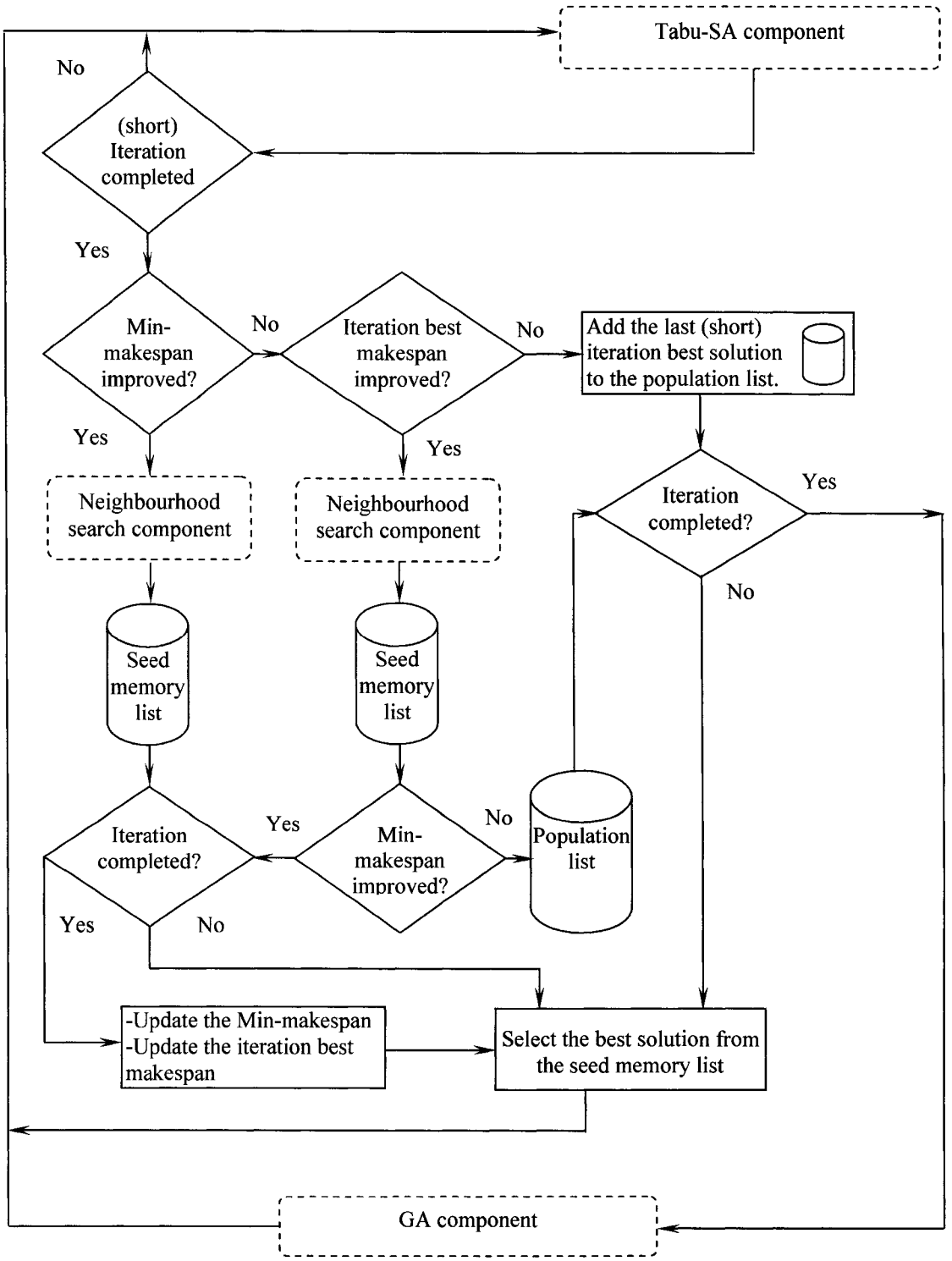


Figure 6-4. Flowchart related to the new SAMED-LB components

list. Similar to the case described above, if the iteration is completed then, the GA module will be activated.

The use of the local search at the end of each (short) iteration is subjected to improvement in the overall or the iteration's best solution. Advances in the iteration best solution without affecting the overall best solution generally occurs after a new initial solution is introduced.

Applying the local search whenever there has been improvement in the overall best solution may increase the chance of finding even better solutions as the area under investigation is likely to be productive. The purpose of employing the local search when a relative improvement (*i.e.,* improvement in iteration's best solution) has been realized is twofold: it helps to rapidly improve the attribute of the new initial solution and assist in improving the overall best solution as well.

The temporal complexity of the SAMED-LB can be calculated follows:

$O(((mn)^3(\ln n)+(qmn/S_{ms}))L_{ms}\times S_{ms})+P^{GA}O(L_{ms}mn)$ per long iteration

where m and n represent the number of machines and number of jobs respectively, q is the number of neighbours on the critical path, S_{ms} is the size of the short-term memory, L_{ms} is the size of the long-term memory, and P^{GA} is the probability that GA component is called.

Similar to that of the SAMED-JSS algorithm, the second term in the above expression is clearly of lower complexity than the first term, therefore the overall complexity of the SAMED-LB is in the order of:

$$O((mn)^3(\ln n)(L_{ms}S_{ms})+(qmn)L_{ms}) \tag{6.11}$$

Detailed analysis of the SAMED-LB temporal complexity is presented in Appendix D.

6.2 Discussion and computational results

6.2.1 The effect of SAMED components on the search performance

The impact of short-term memory in improving the performance of the conventional simulated annealing has been investigated in several studies (Osman 1993, Zolfaghari and. Liang 1999, El-Bouri *et al.,* 2007). In this study however, the effect of long-term

memory along with the GA component on the search performance is investigated. For this purpose, in the SAMED-JSS algorithm the long-term memory and the GA components are replaced with a random solution generator. This algorithm is named SA with memory and multi-start capability. The algorithm performs quite similar to the SAMED-JSS. However, instead of diversifying the search using one of the newly generated offspring in the new population, the algorithm reinitiates the search by employing randomly generated new solutions.

To compare the performance of the two algorithms (i.e., SAMED-JSS and SA with memory and multi-start capability), nine hard job shop scheduling problems have been selected from literature. The two algorithms are coded using Visual Basic and the experiment is run on X86 based PC with 697 MHz CPU. The computational time is set to 300 seconds for MT10 (Fisher and Thompson, 1963), MT20 (Fisher and Thompson, 1963), LA20 (Lawrence, 1984), LA21, and LA22 (Lawrence, 1984), and 420 seconds for ABZ7 (Adams *et al.* 1988), ABZ8 (Adams *et al.* 1988), LA25 as well as LA29 (Lawrence, 1984). The algorithms were run 20 times from randomly

Table 6-1. Computational results for the analysis of SAMED-JSS components effects

Problem	Size ($n \times m$)[a]	SAMED		SA with memory & multi-start capability		SA with memory [b]
		Best solution	Average	Best solution	Average	Best solution
MT10	10 × 10	930	987.8	971	1022.9	947
MT20	20 × 5	1173	1191.7	1195	1286	1197
ABZ7	20 × 15	690	714.1	717	730.9	-
ABZ8	20 × 15	692	709.3	710	731.2	-
LA20	10 × 10	909	927.9	918	942.8	-
LA21	15 × 10	1055	1094.4	1081	1121.3	1081
LA22	15 × 10	937	972.8	963	994.3	-
LA25	15 × 10	985	1020.2	1019	1049.9	-
LA29	20 × 10	1219	1052.1	1269	1306.6	1338

a) $n \times m$ = n jobs and m machines; b) El-Bouri *et al.* (2007)

generated initial solutions. The size of the memories (including both population size and short-term memories) is fixed at 10 for MT20 and 20 for the remaining problems. The computational results are summarized in Table 6-1. According to the results presented in

this table, for all test problems the SAMED-JSS provide solutions with shorter makespan and lower average. This clearly indicates the importance of the long-term memory and the GA component in the SAMED-JSS. In Table 6-1, the computational results pertaining to SAMED-JSS and SA with memory and multi-start capability have been also compared with those provided by El-Bouri *et al.* (2007).

6.2.2 Computational results

To evaluate the performance of the proposed metaheuristics, the algorithms have been coded in Visual Basic and tested using 37 well-known classical job shop scheduling problems selected from the literature. The experiments were run on an X86 based PC with 697 MHz CPU. In SAMED-JSS, the memory size is set to 10 for small problems and 20 for medium and large problems. Here, small problems are defined as $10 \leq n \leq 20$ and $m = 5$, medium as $n = m = 10$, and large as $10 < n \leq 30$ and $10 \leq m$, respectively (n is the number of jobs and m represents the number of machines). The value of parameter Ω, the diversification coefficient, is set to 1 for all of the benchmark problems. The algorithm was run several times (five runs for small problems and 20 runs for the rest of the problems) from randomly generated initial solutions.

Table 6-2 summarizes the computational results of the 37 problems obtained using SAMED-JSS and SAMED-LB. In the table, the names of the problems have been taken directly from the literature. They stand for the first letters of the last names of authors who presented the problems (e.g., MT: Muth and Thompson, ABZ: Adams, Balas, and Zawack). The results presented in this table correspond to the best makespan obtained over several runs and associated computing time for each problem. Among the 37 instances, the SAMED-JSS found the optimal solution in 29 cases (78.4%) within reasonably low computational times. For the other eight instances, near optimal solutions were found (with maximum 5% deviation from the best known solution in problems ABZ7 and LA29). When an optimal solution was obtained, the search time was recorded and the search was terminated. Otherwise, the run times were fixed ranging from five seconds to seven minutes (420 seconds) depending on the size and complexity of the problem. The computational time (seconds) is 5 for LA05, LA06, and LA09-LA14 (Lawrence, 1984); 10 for LA1, LA2, LA4, LA7, LA8, and LA15 (Lawrence, 1984); 90

for LA31, LA33, and LA35 (Lawrence, 1984); 180 for LA16-LA19 (Lawrence, 1984), LA23 (Lawrence, 1984), ABZ5, and ABZ6 (Adams *et al.* 1988); 200 for LA30 and LA34 (Lawrence, 1984); 300 for MT10, MT20, LA20 (Fisher and Thompson, 1963), LA21, LA22, LA26, and LA29 (Lawrence, 1984); and 420 for ABZ7, ABZ8 (Adams *et al.* 1988), and LA25 (Lawrence, 1984), respectively.

In Table 6-2, the two proposed metaheuristics are also compared with seven well-known algorithms selected from the literature. Tabu search algorithm proposed by Nowicki and Smutnicki (1996) is still considered as one of the most efficient techniques for the job shop scheduling problem. The other selected methods include the two hybrid algorithms proposed by Gonçalves *et al.* (2005) and Wang and Zheng (2001), a simulated annealing algorithm (Van Laarhoven *et al.,* 1992), a genetic algorithm (Croce *et al.*, 1995), the GRASP (Binato *et a.l*, 2002), and the GRASP with path-relinking (Aiex *et al.*, 2003).

In the table, numbers in parentheses represent the time (sec.) at which the best solution was found. The results presented in this table clearly indicate that the SAMED-JSS algorithm outperforms other algorithms in terms of solution quality and/or computational time in most cases. It is worthwhile to mention that since the algorithms have been run on non-identical computers (i.e., the computers used by other authors could be either slower or faster than the one used in this study), it would be difficult to compare or to comment on the computational efficiency of the algorithms.

Table 6-2 also presents the computational results obtained by the SAMED-LB. In the SAMED-LB, the tabu list size varied from 5 to 20 depending on problem size and complexity. The population size was set to 30 for the last four problems (LA31 to LA35) and 20 for the others. The coefficient γ was set to unity for LA1 to LA5 and 1/3 for the rest. The computational time for each problem was fixed in the same range as that used in the SAMED-JSS. The results presented in Table 6-2 clearly indicate that the two new features, i.e., local search and blockage removal, have significantly improved the algorithm performance for this particular application. The SAMED-LB finds the optimal solutions for 31 out of the 37 test problems (84%) with less effort compared to other algorithms. In fact, the SAMED-LB has outperformed SAMED-JSS in 35 out of 37 cases in terms of solution quality or computing time, or both. The makespan of the seven test

problems, MT20, ABZ7, ABZ8, LA20, LA22, LA25, and LA29 have been improved and among them two are optimal. In 21 test problems, the SAMED-LB was able to find the optimal solutions with 16% to 90% shorter computing time as shown in Figure 6-5. Only in two test problems (LA19 and LA26), the same optimal solutions were obtained in slightly longer computational times.

The performance of the proposed methods, i.e., the SAMED-JSS and the SAMED-LB, have also been evaluated and compared using eight very large sized test problems. The problems were generated randomly and named as RGP (Randomly Generated Problem). In the SAMED-LB, the size of the tabu and population list is set to 10 and 30 respectively. The size of both short and long-term memories in the SAMED-JSS is set to 30. The computational times are 900 seconds for problems with 100 jobs and 20 machines and 600 seconds for problems with 70 jobs and 20 machines. The experiments were performed on a Pentium 4 PC with 3.4 GHz CPU. Each method has been run five times from a randomly generated initial solution. For each method and test problem, the best makespan obtained over five runs has been reported in Table 6-3. According to the computational results presented in this table in seven out of eight RGPs the solutions found by the SAMED-LB are of better quality than those obtained by the SAMED-JSS.

Furthermore, the convergence behaviour of the SAMED-JSS algorithm is compared with those of the conventional SA and GA. For fair comparison, the same encoding scheme, neighbourhood structure, and cost function are used for all the three algorithms. In GA, the size of the population is set to 100 and the probability of crossover and mutation are respectively set at 0.85 and 0.1. The performance of the methods on the test problems MT10, LA21, LA26, and LA34 is depicted in Figure 6-6(a)-(d). The SA and SAMED-JSS algorithms have been run from the same initial solutions for the four test problems. The cost values of the initial solutions are 1326, 1923, 2221, and 3190, respectively. Initial populations for the genetic algorithm have been generated randomly.

It is also important to note that, in the case of SAMED-JSS, the most of the cost reduction in all cases of Figure 6-6 was achieved in the first 10 seconds whereas for the other methods the cost reduction process prolonged to very late stages. This observation is particularly important when dealing with very large problems. With the SAMED-JSS

algorithm, very good solutions can be achieved in a reasonably short period of time without waiting for the sluggish improvement present in other methods.

Moreover, the performance of the two proposed algorithms on the randomly generated problems is highlighted by comparing with a conventional GA and a conventional SA as shown in Figure 6-7. The cost values of the best member of the populations for the four problems are 1436, 1654, 2035, and 2852, respectively. As shown in these Figures, the SAMED-JSS improves the solution much faster, particularly at the early stage, than the conventional SA and GA.

Table 6-2. Computational results and comparison with other algorithms

Problem	Size ($n \times m$)[a]	Van Laarhoven *et al.* 1992	Croce *et al.* 1995	Nowicki and Smutnicki 1996	Wang and Zheng 2001	Binato *et al.* 2002	Aiex *et al.* 2003	Gonçalves *et al.* 2005	Optimal	SAMED-JSS	SAMED-LB
MT10	10 × 10	930 (57772)	946 (628)	930	930 (44)	938	930 (10125)	930 (292)	930[b]	**930**(168)[c]	**930**(50)
MT20	20 × 5	1165 (62759)	1178 (675)	1165	1165 (90)	1169	1165 (209160)	1165 (204)	1165	1173(240)	**1165**(176)
ABZ5	10 × 10	-	-	1234	-	1238	1234 (2530)	-	1234	**1234**(91)	**1234**(43)
ABZ6	10 × 10	-	-	943	-	947	943 (678)	-	943	**943**(38)	**943**(20)
ABZ7	20 × 15	-	-	-	-	723	692 (583000)		656	690(373)	687(412)
ABZ8	20 × 15	-	-	-	-	729	705 (497800)	-	627-670	695(249)	692(315)
LA01	10 × 5	666 (20)	666 (282)	666	666 (6)	666	666 (<1)	666 (37)	666	**666**(2)	**666**(<1)
LA02	10 × 5	655 (24)	-	655	-	655	655 (10.9)	655 (51)	655	**655**(3)	**655**(1)
LA03	10 × 5	606 (129)	-	597	-	604	597 (29.5)	597 (39)	597	**597**(13)	**597**(3)
LA04	10 × 5	590 (121)	-	590	-	590	590 (15.7)	590 (42)	590	**590**(1)	**590**(1)
LA05	10 × 5	593 (5)	-	593	-	593	593 (<1)	593 (32)	593	**593**(<1)	**593**(<1)
LA06	15 × 5	926 (16)	926 (473)	926	926 (12)	926	926 (<1)	926 (99)	926	**926**(<1)	**926**(<1)
LA07	15 × 5	890 (66)	-	890	-	890	890 (<1)	890 (86)	890	**890**(1)	**890**(1)
LA08	15 × 5	863 (16)	-	863	-	863	863 (1.6)	863 (99)	863	**863**(<1)	**863**(<1)
LA09	15 × 5	951 (13)	-	951	-	951	951 (<1)	951 (94)	951	**951**(<1)	**951**(<1)
LA10	15 × 5	958 (14)	-	958	-	958	958 (1.15)	958 (91)	958	**958**(<1)	**958**(<1)
LA11	20 × 5	1222 (32)	1222 (717)	1222	1222 (26)	1222	1222 (<1)	1222 (197)	1222	**1222**(1)	**1222**(<1)
LA12	20 × 5	1039 (34)	-	1039	-	1039	1039 (<1)	1039 (201)	1039	**1039**(<1)	**1039**(<1)
LA13	20 × 5	1150 (32)	-	1150	-	1150	1150 (<1)	1150 (189)	1150	**1150**(<1)	**1150**(<1)
LA14	20 × 5	1292 (27)	-	1292	-	1292	1292 (<1)	1292 (187)	1292	**1292**(<1)	**1292**(<1)
LA15	20 × 5	1207 (34)	-	1207	-	1207	1207 (24.5)	1207 (187)	1207	**1207**(3)	**1207**(1)
LA16	10 × 10	956 (686)	979 (637)	945	945 (29)	946	945 (2951)	945 (232)	945	**945**(68)	**945**(48)
LA17	10 × 10	784 (112)	-	784	-	784	784 (65.8)	784 (216)	784	**784**(30)	**784**(18)
LA18	10 × 10	861 (112)	-	848	-	848	848 (453.5)	848 (219)	848	**848**(84)	**848**(14)
LA19	10 × 10	848 (830)	-	842	-	842	842 (302.8)	842 (235)	842	**842**(36)	**842**(42)
LA20	10 × 10	902 (667)	-	902	-	907	902 (27710)	907 (235)	902	909(289)	**902**(195)
LA21	15 × 10	1063 (1991)	1097 (1062)	1047	1058 (184)	1091	1057 (351000)	1046 (602)	1040-1053	1055(264)	1055(224)
LA22	15 × 10	938 (2163)	-	927	-	960	927 (89700)	935 (594)	927	937(149)	930(348)
LA23	15 × 10	1032 (275)	-	1032	-	1032	1032 (393.9)	1032 (598)	1032	**1032**(92)	**1032**(9)
LA25	15 × 10	992 (2133)	-	977	-	1028	984 (105280)	986 (609)	977	985(399)	984(216)
LA26	20 × 10	1218 (4342)	1231 (1545)	1218	1218 (418)	1271	1218 (22225)	1218 (1388)	1218	**1218**(125)	**1218**(141)
LA29	20 × 10	1218 (4408)	-	1160	-	1293	1203 (308500)	1196 (1350)	1120-1195	1219(259)	1204(285)
LA30	20 × 10	1355 (3956)	-	1355	-	1368	1355 (22830)	1355 (1260)	1355	**1355**(91)	**1355**(48)
LA31	30 × 10	1784 (1517)	1784 (2762)	1784	1784 (546)	1784	1784 (2676)	1784 (3745)	1784	**1784**(37)	**1784**(12)
LA33	30 × 10	1719 (1880)	-	1719	-	1719	1719 (875.3)	1719 (3637)	1719	**1719**(25)	**1719**(16)
LA34	30 × 10	1721 (1886)	-	1721	-	1753	1721 (4016.5)	1721 (3615)	1721	**1721**(128)	**1721**(47)
LA35	30 × 10	1888 (434)	-	1888	-	1888	1888 (3483.2)	1888 (3716)	1888	**1888**(42)	**1888**(25)

a) $n \times m$ = n jobs and m machines; b) Numbers in column three to 12 represent the makespans (**bold** = optimal solution), c) computing times (sec.).

Table 6-3. Computational results for randomly generated problems

Problem	Size($n \times m$)[a]	SAMED-JSS[b]	SAMED-LB
RGP-1	70×20	9016	8942
RGP-2	70×20	8265	8232
RGP-3	70×20	8363	8309
RGP-4	100×20	11163	10987
RGP-5	100×20	10898	10922
RGP-6	100×20	10950	10831
RGP-7	100×20	12233	12206
RGP-8	100×20	11039	10970

a) $n \times m = n$ jobs and m machines; b) Best makespan of five runs

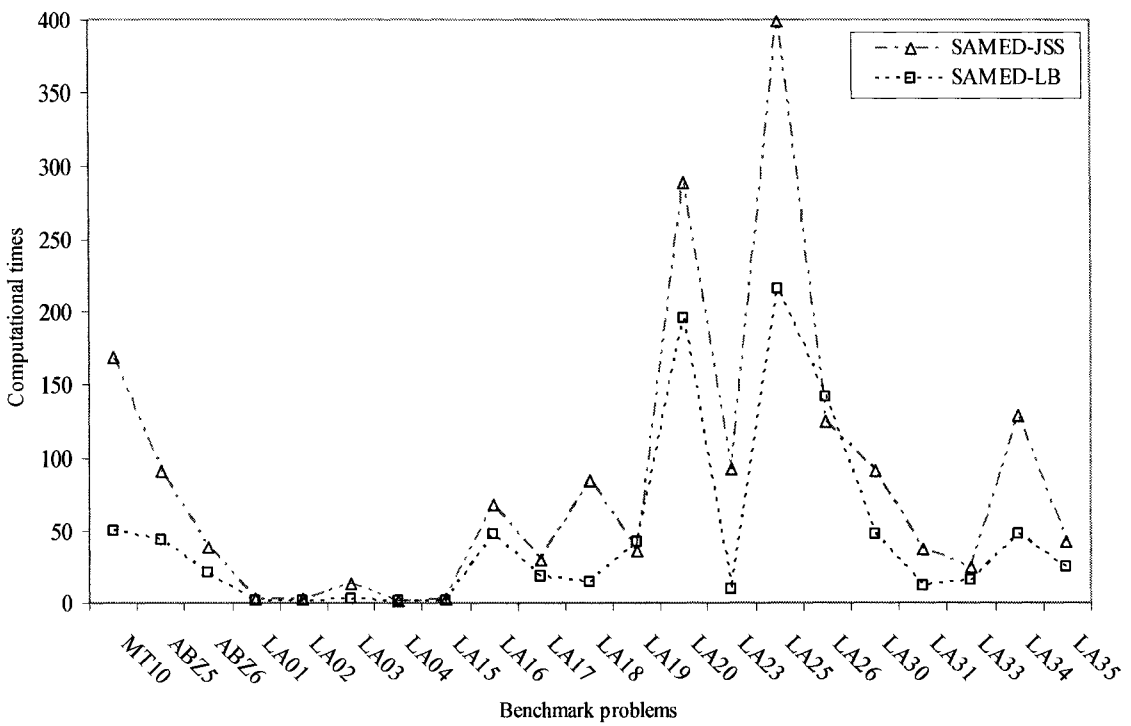
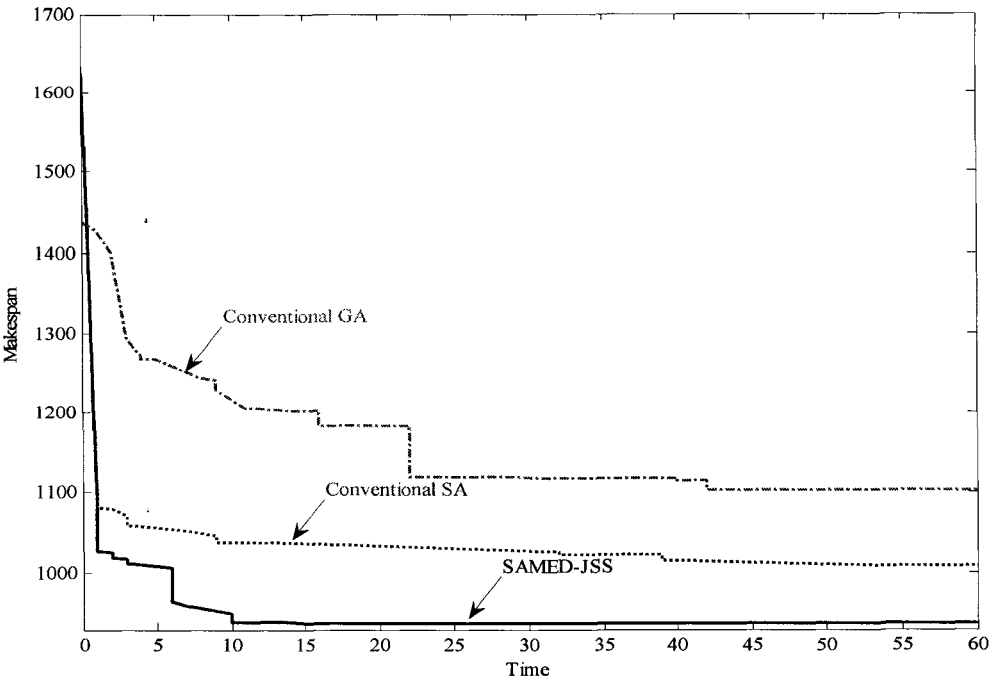
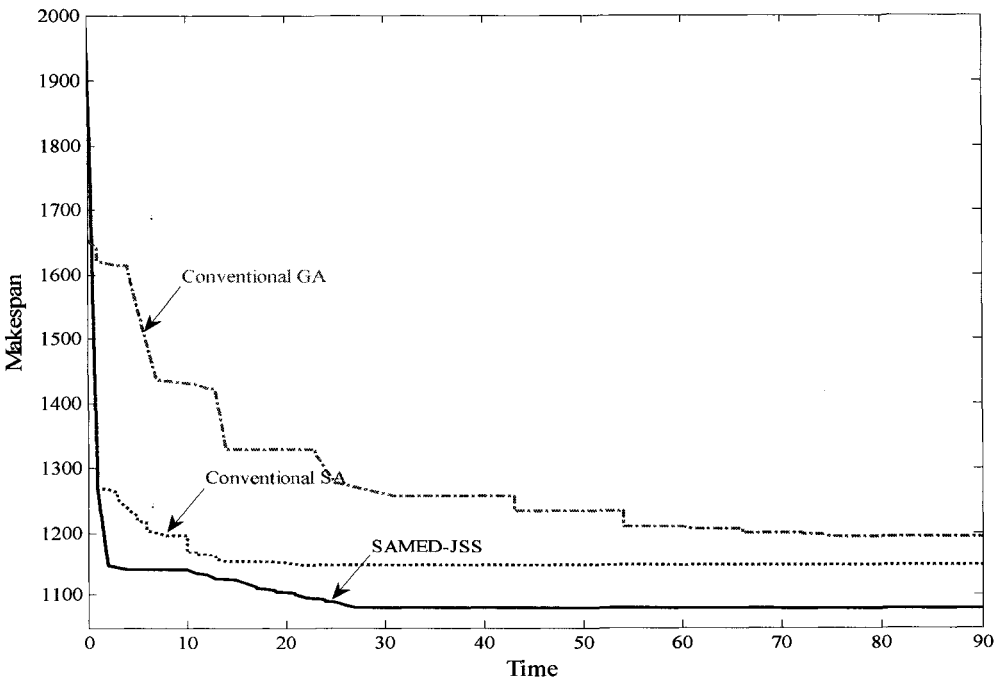


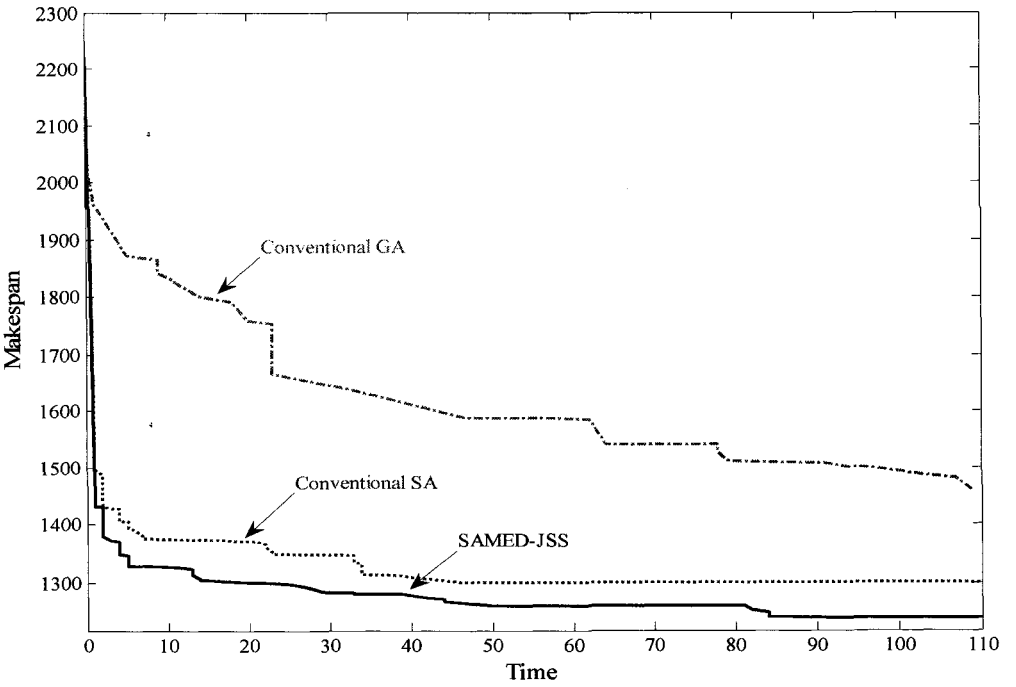
Figure 6-5.Comparision of computational times



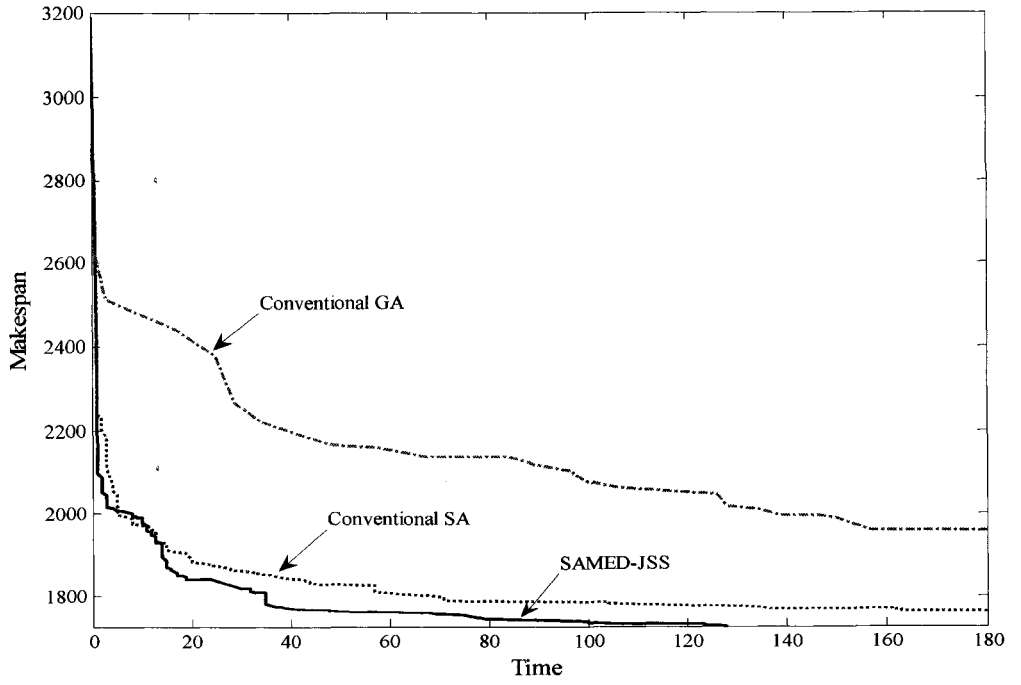
(a)Test problem MT10



(b)Test problem LA21

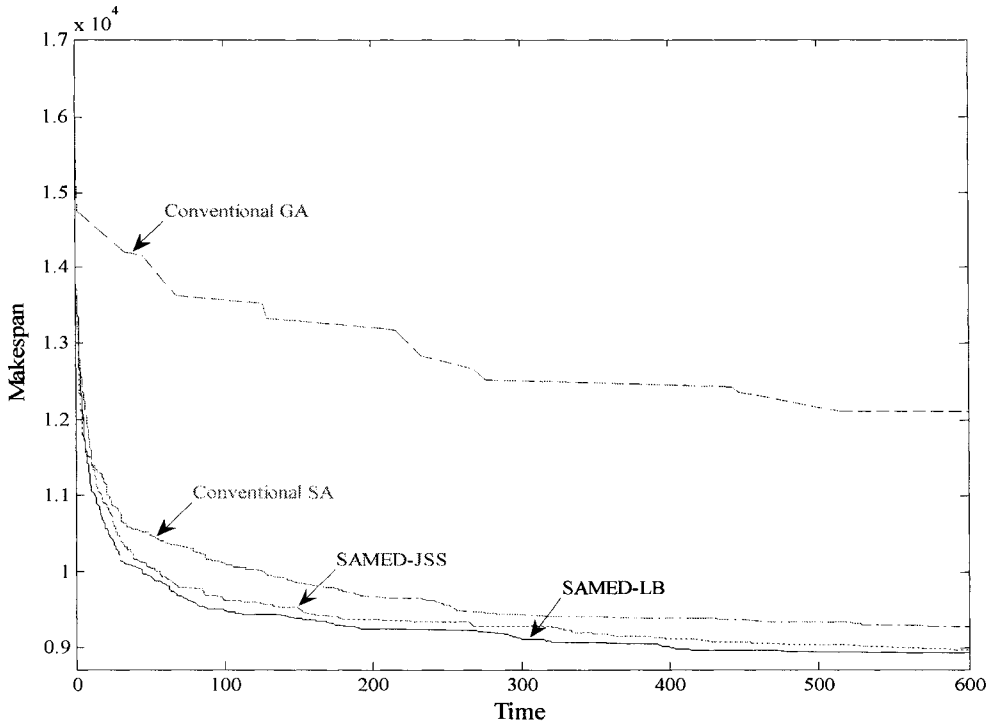


(c)Test problem LA26

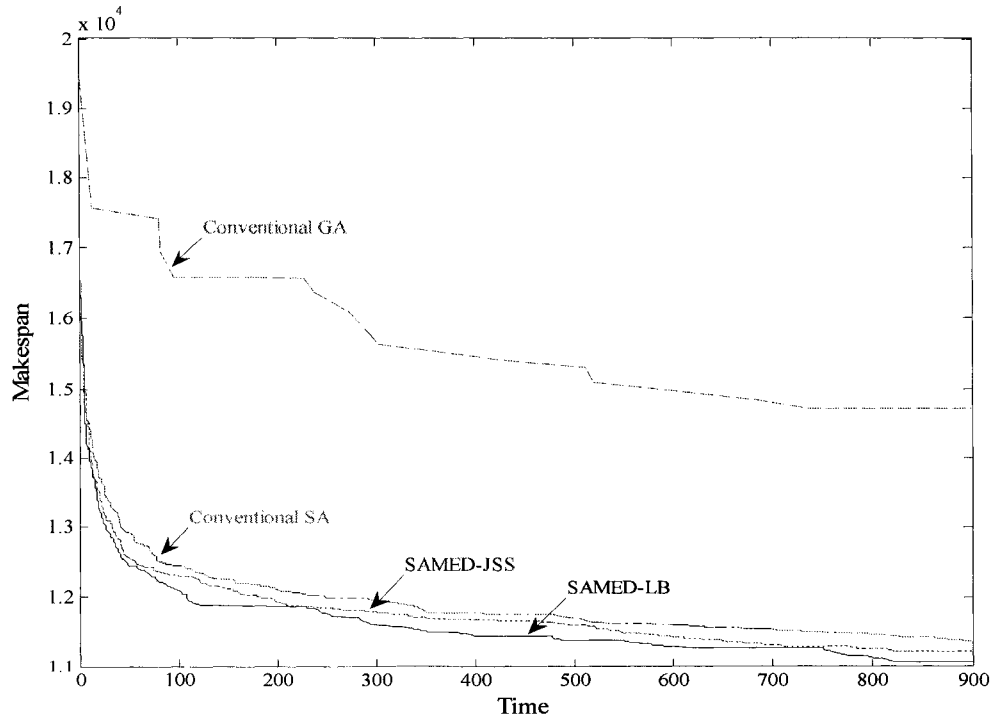


(d) Test problem LA34

Figure 6-6.Convergence comparison of the SAMED-JSS with conventional simulated annealing and genetic algorithm



(a) Test problem RGP-1



(b) Test problem RGP-4

Figure 6-7.Convergence comparison of the proposed heuristics with conventional simulated annealing and genetic algorithm

6.3 Application of the SAMED to flow shop scheduling problem

Based on the generic framework of the SAMED presented in chapter 5, a search algorithm (SAMED-FSS) has been developed and applied to the flow shop scheduling (FSS) problem. In the following sub-sections, the flow shop scheduling problem is described followed by specification of the SAMED-FSS components.

6.3.1 Flow shop scheduling problem

The permutation flow shop scheduling problem studied in this study can be described as follows. Given a set of machines (M_1, M_2, M_3...,M_m) and a number of jobs (J_1, J_2, J_3,...,J_n), each job has to be processed on m machines in the order specified by the indices of the machines. Each job consists of a sequence of m operations, O_{1j}, O_{2j}, ..., O_{ij},...,O_{mj}, each of them corresponding to the processing of job j on machine i during an uninterrupted processing time P_{ij}. Each machine can only process one job at a time and it is assumed that each machine processes the jobs in the same order. The objective is to find the schedule that has the minimum makespan (the duration in which all jobs are completed). The problem is known to be NP-complete for three or more machines (Garey *et al.*, 1976). Therefore, the research has been concentrated mainly on developing heuristics to obtain near optimal solutions for the problem.

6.3.2 The SA module

The SA module is the core of the SAMED-FSS. It handles several key tasks including neighbouring solution generation, solution evaluation, and solution acceptance or rejection decisions. In the SA module, a solution to a flow shop scheduling problem is represented by a string composed of several elements each corresponding to a job number (job based scheme). The sequence of the job numbers on the string indicates the processing order of jobs on all machines. For instance, in a flow shop scheduling problem with nine jobs, a solution could be represented as follows

1 3 5 4 2 7 6 8 9

Furthermore, two neighbourhood generation schemes, pairwise exchange and insertion method, are employed to generate a neighbouring solution. Pairwise exchange mechanism generates a neighbouring solution by swapping the position of two randomly selected genes on a chromosome. To generate a neighbouring solution by insertion technique, a randomly selected gene is inserted in front or after another gene on the chromosome. Moreover, the following cooling schedule is adopted:

$$\theta_{l+1} = \frac{\theta_1}{1 + \lambda \ln(L)} \tag{6.12}$$

where θ_1 is the initial temperature, L represents the move number, and λ is a control parameter. With this cooling schedule, the temperature declines gradually as the search progresses and the number of moves increases. The rate of the temperature reduction could be controlled by the parameter λ.

Finally, the cost function of a given solution is the makespan of the schedule, which will be used to evaluate the solution quality.

6.3.3 The GA component

The two evolutionary operators known as C1 and C2 (Reeves, 1995) (also called one- and two- point crossover operators, respectively) have been widely used by researchers to recombine the chromosomes in GAs. According to the experiments conducted by Reeves (1995), C2 operator may cause excessive disruption in chromosomes, resulting in slow convergence. For this reason, the GA component of the SAMED-FSS algorithm adopted the C1 crossover operator and a simple selection mechanism. To generate a new population first, two parents (e.g., Parent 1 and Parent 2) are selected randomly from the population list one at a time. Then, the C1 crossover operator is used to generate the new population members. To generate new offspring by C1, a crossover point is first selected randomly on one parent, e.g., Parent 1. Then all of the elements (*i.e.,* genes) on the left side of the crossover point are taken from Parent 1. Afterwards, the offspring chromosome is completed by taking the *legitimate* elements (i.e., the elements of parent 2 that are not identical to those already taken from parent 1) from Parent 2 sequentially.

The second solution is generated by selecting a crossover point on Parent 2 and repeating the same steps described for Parent 1.

6.3.4 Blockage removal

The aim of using tabu list in the SAMED-FSS is twofold. First, it reduces the size of the solution space. This may help to find better solutions in short computational time. Secondly, it avoids revisiting solutions that have been recently explored. Nonetheless, a tabu list may have some side effects that could deteriorate the performance of the search. For instance, such a list can limit the number of potentially acceptable solutions for the SA component, thereby causing a deadlock situation. Especially at low temperatures, the chance of being trapped in such an undesirable situation increases. Hence as a remedy, a blockage removal feature is added to the metaheuristic. With this feature, the last examined solution is accepted regardless of the cost whenever the number of trials conducted by the SA component to find an acceptable solution exceeds the value specified by the following formula:

$$\text{Maximum number of unacceptable moves} = \gamma \times [(n - TabuSize)((n - TabuSize) - 1)]/2 \quad (6.13)$$

where n and $TabuSize$ denote the number of jobs and the size of the tabu list respectively. Parameter γ is a coefficient that controls the magnitude of the maximum (allowable) number of unacceptable moves.

In a simple pairwise exchange neighbourhood generation scheme, $n\times(n\text{-}1)/2$ neighbouring solutions could be generated by swapping the position of two randomly selected jobs on a string. However, the number of potential neighbouring solutions decreases as the search continues and more jobs are added into the tabu list. When the tabu list reaches its predetermined size, the number of (neighbouring) solutions is reduced to $(n\text{-}TabuSize)\times[(n\text{-}TabuSize)\text{-}1]/2$ solutions. This value is regarded as the maximum number of (allowable) trials to find an acceptable solution.

6.3.5 The search procedure

The details of the SAMED-FSS search procedure is presented in the following pseudo code format.

Initialization

1. Generate an initial solution, S^{ini},
2. Substitute the current solution, S^{cu}, with the initial solution, S^{ini},
3. Calculate the makespan (C^{cu}) of the current solution, S^{cu},
4. Set:
   a) Short-term memory size (Ssize)
   b) Long-term memory size (Lsize)
   c) Coefficient of the maximum number of unacceptable moves (γ)
   d) $C^{min} = C^{cu}$
   e) Scount (short iteration counter) =1
   f) Lcount (long iteration counter) = 1

Do While STOP criterion met
{
Do While Lcount < Lsize + 1 (long iteration completed)
{
Do While Scount < Ssize + 1 (short iteration completed)
{
NumberOfTry = 1
Do While solution accepted
{
Generate a candidate solution, S^{ca}, using pairwise exchange scheme
IF either of the two selected jobs is in the tabu list **THEN**
Generate another solution
ELSE
IF candidate solution is rejected **THEN**
NumberOfTry = NumberOfTry + 1
IF NumberOfTry > MNUM* **THEN**
Accept S^{ca}
END IF
END IF
Update: C^{min} and the makespan of the Short Iteration Best Solution (SIBS), C^{SIBS} (Scount)
Scount = Scount +1
END IF
}
Loop
Update both the tabu list and seed memory list and move to S^{ca}
}
Loop
If Lcount = 1 **THEN**
Replace S^{cu} with the best solution in the seed memory list and empty both tabu and seed memory lists
Update Long Iteration Best Solution (LIBS), C^{LIBS}(Lcount)
Lcount = Lcount +1
ELSE
IF C^{SIBS} (Scount) < C^{LIBS}(Lcount-1) **THEN**

IF long iteration is completed **THEN**
Update C^{LIBS}(Lcount)
Replace S^{cu} with the best solution in the seed memory list and empty both tabu and seed memory lists

ELSE
Replace S^{cu} with the best solution in the seed memory list
END IF
ELSE
Add the best solution from the seed memory list into the population list
IF long iteration is NOT completed **THEN**
Replace S^{cu} with the best solution in the seed memory list
ELSE
Generate a new population using solutions stored in the population list and the crossover operator in the GA component.
Determine the makespan ($C^{popBest}$) of the best member of the population, $S^{popBest}$.
IF $C^{popBest} < C^{min}$ **THEN**
Replace S^{cu} with $S^{popBest}$
ELSE
Generate a random number between zero and one, Rnd
IF Rnd ≥ current value of the CT function **THEN**
Replace S^{cu} with the best solution in the seed memory list
and empty both tabu and seed memory lists
ELSE
Scan the new population and select the first offspring whose makespan is different from that of
the iteration's best solution
END IF
END IF
END IF
END IF
END IF
}
Loop
}
Loop

*: Maximum Number of Unacceptable Moves

In the above algorithm, a new population is generated by applying the C1 crossover operator to the solutions stored in the population list. Each time the crossover combines a pair of parent chromosomes, two offspring are generated. The makespans of the newly generated offspring are evaluated and compared and the one that possesses shorter makespan is added to the new population. The other offspring is simply discarded.

The chromosomes of the newly generated offspring carry elements from the previously visited iteration's best solutions. In situations when there is evidence that the search might have reached a local optimum, substituting the current solution with a new initial solution is an option to escape the trap. Selecting such a (initial) solution form the group of newly created offspring not only helps the search to stay away from the local optima, it may also speed up the process of upgrading the overall best solution. However, due to genetic similarities between the offspring and the iteration's best solutions, there is also a possibility that the search may quickly return to the same local optima (i.e., the best solution found so far). To reduce the chance of revisiting the same best solution, whenever the search restarts from a new initial solution the temperature is reset to a high level. A high level of temperature makes it possible for the search to explore other parts of the solution space (diversification). Furthermore, given the relationship between the temperature function and its complement (CT function), raising the level of temperature immediately lowers the value of CT function. A low CT function blocks the reallocation of the initial solution prematurely (intensification). This could be regarded as the second result of the temperature change at the beginning of a new iteration where the search restarts from a new initial solution.

With the temperature function used in the SA component, the CT function is presented as

$$\text{CT function} = \Omega \times \left(1 - \left(\frac{\theta_i}{\theta_1}\right)\right) \tag{6.14}$$

where θ_i is the temperature at i^{th} move (transition), θ_1 is the initial temperature, and Ω is the diversification coefficient.

6.3.6 Application of stand-alone SA and GA, as well as GA with local search for the flow shop scheduling

For comparison, algorithms based on stand-alone SA and GA, and hybrid GA with local search (GA-LS) have been developed. Some of the elements of these algorithms are described as follows.

6.3.6.1 Stand-alone GA and GA-LS algorithm

In both the stand-alone GA and GA-LS algorithms, a job based scheme has been used to represent a solution. According to this encoding scheme, each solution is represented by a string of job numbers. The sequence of job numbers in the string indicates the order in which jobs are to be processed on each machine.

To generate new offspring, the current population is sorted in descending order of makespan. Then two parents are selected according to the following probability distribution function proposed by Reeves (1995):

$$P([kc]) = \frac{2kc}{PopSize(PopSize+1)} \tag{6.15}$$

where $[kc]$ is the kc^{th} chromosome in descending order of makespan and $PopSize$ is the population size.

Referring to Figure 6-9, once the two parents are selected and, if the probability of crossover is greater than a random number (between zero and one), the C1 crossover operator is used to generate two offspring. For each child, the probability of mutation is compared to a random number. This leads to the following two cases:

a) If the mutation probability is greater than the random number, a local search (LS) module is used as the mutation operator as explained below. To perform the mutation, Q genes from the offspring chromosome are selected randomly. Then, a set of neighbouring chromosomes is generated by swapping the positions of the selected genes. To generate a neighbouring chromosome, the offspring chromosome is scanned from left to right and the position of each gene is consecutively exchanged with one of the selected genes on the right hand side of the first gene (Cheng, 1997). The quality of the offspring with regard to the resulting schedule length is tested after each swap. In the end, the best swap producing the shortest schedule is made. The corresponding makespans of the offspring

chromosome before and after the mutation operation are compared and if the mutation deteriorates the quality of the offspring, the mutation result is revoked and then the chromosome is added to the population.

Accordingly, the number of neighbouring chromosomes of an offspring could be calculated according to the following formula:

$$\text{Number of neighbouring chromosomes} = Q \times (Q-1)/2 \tag{6.16}$$

where Q = min (n, m). For instance, in a problem with 16 jobs and 4 machines (Q=4), an offspring chromosome is generated as follows:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Now, assuming that the four randomly selected genes from left to right are respectively first, fourth, seventh, and twelfth genes, the six potential neighbouring chromosomes are shown in Figure 6-8.

Offspring chromosome	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Neighbour_1	4	2	3	1	5	6	7	8	9	10	11	12	13	14	15	16
Neighbour_2	7	2	3	4	5	6	1	8	9	10	11	12	13	14	15	16
Neighbour_3	12	2	3	4	5	6	7	8	9	10	11	1	13	14	15	16
Neighbour_4	1	2	3	7	5	6	4	8	9	10	11	12	13	14	15	16
Neighbour_5	1	2	3	12	5	6	7	8	9	10	11	4	13	14	15	16
Neighbour_6	1	2	3	4	5	6	12	8	9	10	11	7	13	14	15	16

Figure 6-8. Neighbouring chromosomes of an offspring

b) If the mutation probability is less than the random number, the offspring is added directly to the new population.

If the condition to perform the crossover operator is not met, i.e., the crossover probability is less than the chosen random number, one of the parents is selected randomly and it is treated as a newly generated offspring (Figure 6-9). As soon as the new population is completely generated, those solutions with lower quality in the old

population are replaced with the members of the new population that leads to a shorter makespan.

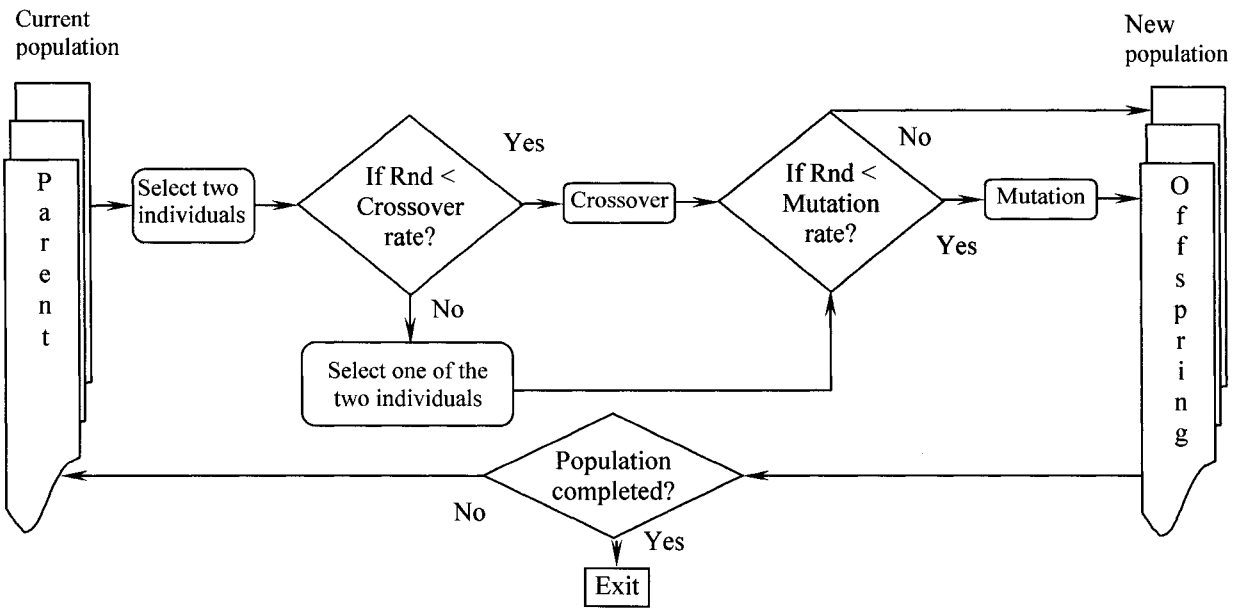


Figure 6-9. Reproduction phase of the GAs

All the features described above are utilized to develop both the stand-alone GA and the GA-LS, except for the above described local search mutation operator which will be used only for the GA-LS. The mutation for the stand-alone GA will be done by simple pairwise exchange. Upon the acceptance of a mutation operator, positions of two randomly selected jobs on a given offspring chromosome are first swapped. The corresponding makespan of the offspring is then calculated and compared to that obtained before the mutation. If improvement is observed, the offspring is added to the new population. Otherwise, the swap is cancelled before the offspring joins the new population.

6.3.6.2 Stand-alone SA algorithm

The ingredients of the stand-alone SA such as neighbourhood structure, cooling schedule, and cost functions are the same as those in the SA component of the SAMED-FSS as described in section 6.3.2.

6.4 Computational results

As stated earlier, two neighbourhood functions, the pairwise exchange and insertion technique, are used to generate a neighbouring solution in SA component of the proposed method. The computational results and comparison pertaining to each neighbourhood generation scheme are presented as follow.

To evaluate the performance of the SAMED-FSS, the algorithms have been coded in Visual Basic and tested using two sets of hard benchmark flow shop scheduling problems proposed by Taillard (1993) and Demirkol *et al.,* (1998). The Visual Basic programs for the algorithms with the pairwise exchange neighbourhood scheme and the insertion technique were run on an X86 based PC with 697 MHz CPU and a Pentium 4 PC with 3.3 GHz CPU respectively. For fair comparisons, identical parameters are used in the stand-alone version of SA (or GA) and the SA (or GA) component in the SAMED-FSS. For example, in both the stand-alone SA and the SA component of the SAMED-FSS, the initial temperature and the control parameter are set to 10 and 2, respectively. In both the stand-alone GA and the GA with local search (GA-LS), the population size, the probability of crossover, and the probability of mutation are respectively set to 60, 0.9, and 0.2. The initial population is generated randomly for each test problem.

6.4.1 The pairwise exchange mechanism

The parameters of SAMED-FSS for both benchmark problem sets are presented in Table 6.4. In the Taillard benchmark problems, the computational time (seconds) is limited to 20 for Ta01-Ta05; 300 for Ta11-Ta13, Ta15, Ta21, Ta24, Ta25, Ta41, Ta42, Ta44, and Ta48; 60 for Ta31, Ta33- Ta35, and Ta61; 120 for Ta65, Ta66, and Ta69; 900 for Ta71-Ta73 and Ta76.

For the benchmark problems from Demirkol *et al.,* (1998), the computational time (seconds) is 300 for DMU01-DMU04, DMU06-DMU09, DMU11-DMU14, and DMU19-DMU19; 420 for DMU21-DMU24, DMU26-DMU29, and DMU31-DMU34.

The value of parameter Ω, the diversification coefficient, is set to 0.5 for all of the benchmark problems. The parameters of the SAMED-FSS were simply set at what seemed to be sensible values after some fairly small-scale experiments.

The two genetic algorithms (stand-alone GA and GA-LS) were run 10 times from randomly generated initial populations. The other two algorithms, the stand-alone SA and the SAMED-FSS, were run 10 to 20 times from randomly generated initial solutions. The computational results for the Taillard's benchmark problems are summarized in Table 6-5. The results presented in this table correspond to the best makespan over several runs and associated computing time (sec.) for each test problem. Among the 28 benchmark problems, the SAMED-FSS found the optimal solution for 18 instances and near optimal solutions for the rest of the problems (with maximum deviation of 1.5 percent from optimal in problem Ta42).

Table 6-4.Parameters of the SAMED-FSS

	Problem size($n \times m$)	Short term memory (size)	Long term memory (size)	γ
Taillard's benchmark problems	20 × 5	3	20	2
	20 × 10	5	30	2
	20 × 20	7	30	2
	50 × 5	5	30	2
	50 × 10	5	30	0.5
	100 × 5	10	30	0.5
	100 × 10	5	100	0.3
Demirkol *et al.* benchmark problems	20 × 15	5	50	2
	20 × 20	5	50	2
	30 × 15	5	50	2
	30 × 20	5	50	1
	40 × 15	5	50	1
	40 × 20	5	50	1
	50 × 15	5	50	1

Note: γ is the coefficient of the maximum number of unacceptable moves

The results in Table 6-5 clearly indicate the superior performance of SAMED-FSS over all the other algorithms including stand-alone SA and GA, and GA-LS.

In Table 6-5, the performance of the SAMED-FSS has been also compared to some well-known heuristics including the PSO (Liao *et al.,* 2007), tabu search algorithms of Taillard (1990) and Ben-Daya and Al-fawzan (1998). The comparison of the results shows that in 23 out of 28 test problems, the quality of solutions obtained by SAMED-

FSS is better than those obtained by PSO. Particularly, the solutions obtained by SAMED-FSS for large size problems (e.g., 100×10 and 50×10) have significantly shorter makespans than those provided by the PSO. The results also show that SAMED-FSS is computationally more efficient than the two tabu search algorithms.

Comparison of the results obtained by the conventional SA and GA with local search indicates that in 20 out of 28 test problems, the solutions obtained by the conventional SA are of better quality than those obtained by GA-LS. The stand-alone GA did not perform well compared to the other techniques.

Demirkol *et al.* (1998) presented a set of challenging benchmark problems for the flow shop scheduling problems. They solved the problems using several methods including dispatching rules and shifting bottleneck procedures. The best solution found by any of the methods was considered as the upper bound of each problem.

The computational results for the Demirkol *et al.* (1998) benchmark problems are presented in Table 6-6. The results in this table correspond to the best makespan over several runs for each test problem and associated computational time (second). Similar to that observed in the first set of benchmark problems, the SAMED-FSS clearly outperforms the other methods as it provides solutions with significantly shorter makespan. Although the conventional SA performed better than the GA-LS in the Taillard's test problems, the same conclusion could not be reached for the Demirkol *et al.* (1998) problems. According to the results presented in Table 6-6, in the majority of the cases the GA-LS obtained solutions with shorter makespan compared to those provided by the conventional SA. In this table, performance of the SAMED-FSS has been also compared to that of PSO (Liao *et al.*, 2007). Based on the computational results, the proposed algorithm outperforms the PSO in 19 out of 28 instances.

Furthermore, the standard deviation of the makespans for each test problem and method is presented in Table 6-7. The results pertaining to both Taillard's and Demirkol's benchmark problems clearly indicate that in most cases the standard deviation of the makespans obtained by the SAMED-FSS is significantly lower than those obtained by the other techniques (Figure 6-10), thus indicating better consistency in results.

Moreover, Taillard (1993) provided 10 instances for each benchmark problem size (e.g., 20×5). Five instances from each problem size presented in Table 6-5 were selected and run five times starting from randomly generated initial solutions. Then, the best five makespans for the same problem size were averaged. The results of the Mean Percentage Deviation (MPD) from Taillard's (mean) upper bounds for SAMED-FSS is summarized in Table 6-8. In this table, the MPDs associated with SAMED-FSS have been also compared to those reported in the literature using several other heuristics. The results clearly indicate that SAMED-FSS significantly outperforms the other three methods including GA, SA, and Ant Colony System (ACS) (with respect to the selected performance criterion (i.e., MPDs from Taillard's upper bound) (Figure 6-11).

For the Demirkol *et al* benchmark problems, such reference values (mean percentage deviations from upper bound) are not available in the accessible literature and hence similar comparison could not be made in this study.

Table 6-5. Computational results for the Taillard's benchmark problems

Size ($n \times m$)	Problem	Lower bound	Optimal	SA: Ogbu & Smith 1990	TS[a]: Taillard 1993	TS : Ben-Daya & Fawzan 1998	SA[b]	GA[c]	GA-LS[d]	PSO: Liao et al. 2007	SAMED-FSS[e]	Deviation from optimal (%)
20 × 5	Ta01	1232	1278	1324(390)	1278	1278(504)	1278(7)	1297(1)	1297(1)	1294	**1278**(4)	0
	Ta02	1290	1359	1368(384)	1359	1359(504)	1359(8)	1360(16)	1366(2)	1360	**1359**(1)	0
	Ta03	1073	1081	1197(390)	1081	1081(504)	1081(14)	1107(8)	1098(2)	1090	**1081**(3)	0
	Ta04	1268	1293	1418(396)	1293	1293(504)	1297(20)	1323(17)	1305(3)	1298	**1293**(9)	0
	Ta05	1198	1235	1312(390)	1235	1235(504)	1235(6)	1250(16)	1245(3)	1235	**1235**(2)	0
20 × 10	Ta11	1448	1582	1743(402)	1582	1582(534)	1607(129)	1631(8)	1594(6)	1604	**1582**(187)	0
	Ta12	1479	1659	1805(402)	1659	1659(534)	1679(119)	1709(54)	1671(11)	1685	**1659**(134)	0
	Ta13	1407	1496	1664(408)	1496	1496(534)	1517(212)	1525(53)	1524(5)	1520	**1496**(225)	0
	Ta15	1325	1419	1547(402)	1419	1419(534)	1445(115)	1438(59)	1439(6)	1427	**1419**(239)	0
20 × 20	Ta21	1911	2297	2485(588)	2297	2297(564)	2332(295)	2329(35)	2331(29)	2319	2302(214)	0.22
	Ta24	1810	2223	2408(582)	2223	2223(564)	2243(13)	2271(43)	2246(29)	2243	**2223**(119)	0
	Ta25	1899	2291	2478(582)	2291	2291(564)	2323(12)	2364(35)	2309(46)	2315	2294(186)	0.13
50 × 5	Ta31	2712	2724	2791(2052)	2724	2724(2940)	2724(29)	2731(8)	2724(6)	2724	**2724**(8)	0
	Ta33	2596	2621	-	2621	-	2622(26)	2657(32)	2646(8)	2621	**2621**(22)	0
	Ta34	2740	2751	-	2751	-	2753(17)	2782(19)	2770(17)	2762	**2751**(23)	0
	Ta35	2837	2863	-	2863	-	2864(7)	2887(4)	2886(29)	2864	**2863**(10)	0
50 × 10	Ta41	2907	2991	3399(2784)	3037	3037(3780)	3063(169)	3140(69)	3094(51)	3080	3030(261)	1.30
	Ta42	2821	2867	-	2911	-	2928(298)	3008(141)	2971(64)	2974	2911(47)	1.53
	Ta44	2968	3063	-	3067	-	3079(289)	3147(158)	3113(44)	3103	3067(171)	0.13
	Ta48	2959	3037	-	3048	-	3054(177)	3153(87)	3097(50)	3085	3046(199)	0.3
100 × 5	Ta61	5437	5493	5674(8802)	5493	5493(12300)	5495(15)	5505(16)	5495(20)	5493	**5493**(39)	0
	Ta65	5195	5250	-	5250	-	5253(54)	5261(26)	5255(106)	5250	**5250**(78)	0
	Ta66	5063	5135	-	5135	-	5137(26)	5160(11)	5146(15)	5137	**5135**(62)	0
	Ta69	5385	5448	-	5448	-	5454(98)	5487(91)	5473(23)	5473	**5448**(87)	0
100 × 10	Ta71	5759	5770	6367(11358)	5776	5776(15660)	5785(497)	5887(82)	5843(294)	5843	5774(770)	0.07
	Ta72	5345	5349	-	5362	-	5384(627)	5459(762)	5406(240)	5400	5362(674)	0.24
	Ta73	5623	5676	-	5679	-	5691(494)	5777(887)	5738(327)	5745	5679(298)	0.05
	Ta76	5246	5303	-	5308	-	5321(643)	5391(817)	5349(142)	5341	5308(683)	0.09

Note: a) Tabu Search; b) Conventional SA; c) Standard GA; d) GA with Local Search; e) makespan (time/sec.), **bold**= optimal solution.

Table 6-6. Computational results for the Demirkol *et al* benchmark problems

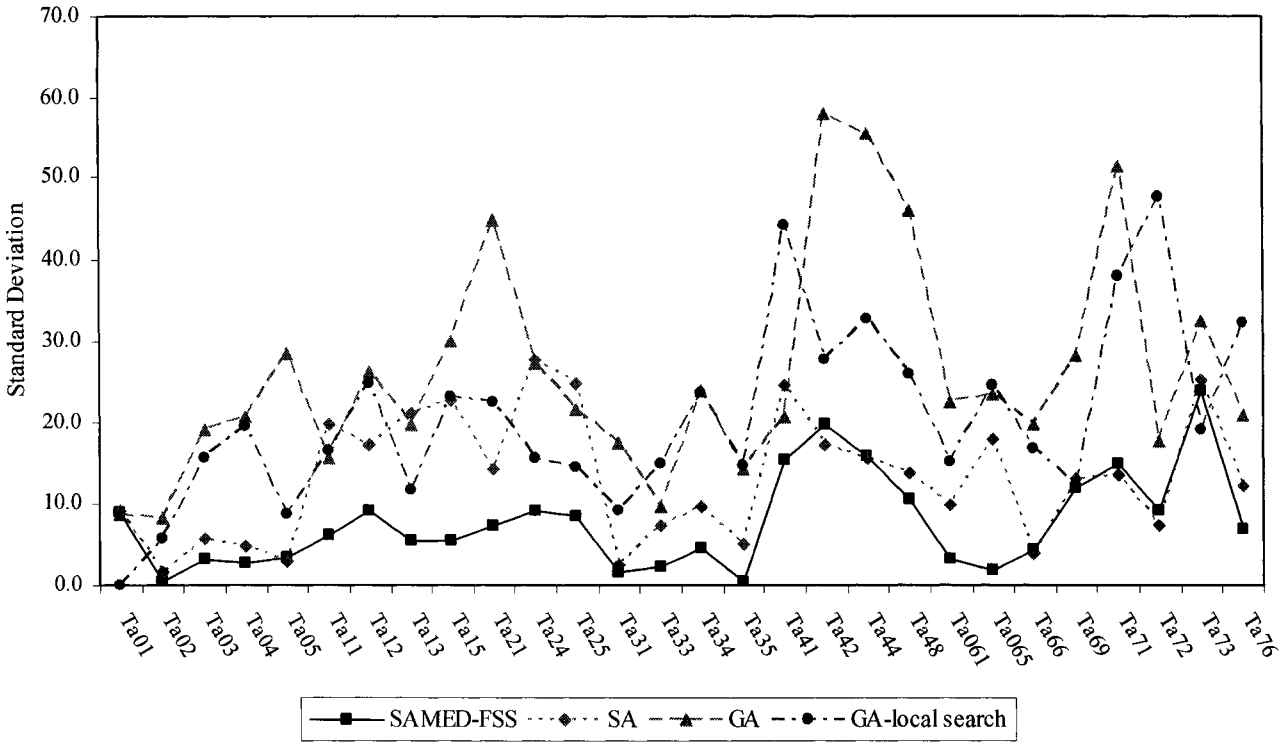
Size ($n \times m$)	Problem	Lower Bound	Demirkol et.al 1998	SA[a]	GA[b]	GA-LS[c]	PSO: Liao *et al.* 2007	SAMED-FSS[e]
20 × 15	DMU01	3354	4437	3984 (34)	3981(10)	3954(19)	3937	***3933***(135)
	DMU02	3168	4144	3851(10)	3855(43)	3816(14)	3571	3792(279)
	DMU03	2997	3779	3590(10)	3633(6)	3566(24)	3981	***3542***(266)
	DMU04	3420	4302	4074(63)	4175(11)	4088(12)	3805	4058(106)
20 × 20	DMU06	3776	4821	4662(98)	4652(11)	4603(49)	4612	***4523***(202)
	DMU07	3758	4779	4552(78)	4552(49)	4479(46)	4570	***4459***(44)
	DMU08	3902	4944	4619(9)	4633(4)	4577(29)	4504	4534(271)
	DMU09	3881	4886	4559(7)	4607(6)	4529(26)	4538	***4505***(133)
30 × 15	DMU11	4020	5226	4655(124)	4703(66)	4654(32)	4658	***4615***(135)
	DMU12	4080	5304	4775(107)	4812(124)	4751(60)	4743	***4709***(150)
	DMU13	4022	5079	4662(54)	4748(138)	4654(40)	4824	***4642***(251)
	DMU14	4490	5605	4906(209)	5016(88)	4929(36)	4928	***4877***(295)
30 × 20	DMU16	4806	6183	5492(48)	5537(171)	5466(85)	5782	***5431***(170)
	DMU17	4772	6037	5829(85)	5924(87)	5786(115)	5485	5748(184)
	DMU18	5004	6241	5815(77)	5870(59)	5815(102)	5486	5815(34)
	DMU19	4899	6095	5613(47)	5637(89)	5565(109)	5848	***5537***(35)
40 × 15	DMU21	5560	6986	6131(36)	6157(293)	6102(103)	6012	6051(326)
	DMU22	5119	6351	5806(60)	5933(281)	5784(89)	6080	***5751***(107)
	DMU23	5290	6506	5964(137)	6024(67)	5959(97)	6173	***5929***(86)
	DMU24	5596	6845	5982(281)	6104(88)	5998(56)	5855	5928(118)
40 × 20	DMU26	5693	7154	6628(245)	6777(217)	6716(159)	6730	***6603***(261)
	DMU27	5998	7528	6788(72)	6911(163)	6869(169)	6723	6769(293)
	DMU28	5990	7469	6945(50)	7086(299)	6933(241)	6973	***6888***(285)
	DMU29	6170	7608	6844(173)	6996(251)	6867(139)	6950	***6856***(164)
50 × 15	DMU31	6290	7673	6795(220)	6875(169)	6792(111)	6725	6772(318)
	DMU32	6355	7679	6724(162)	6857(299)	6750(171)	7143	***6647***(406)
	DMU33	6198	7416	6736(205)	6834(295)	6780(101)	6864	***6658***(241)
	DMU34	6312	7548	6945(223)	7054(417)	6940(144)	7070	***6900***(315)

Note: a) Conventional SA; b) Standard GA; c) GA with Local Search; e) makespan (time/sec.), ***italic bold***= best solution

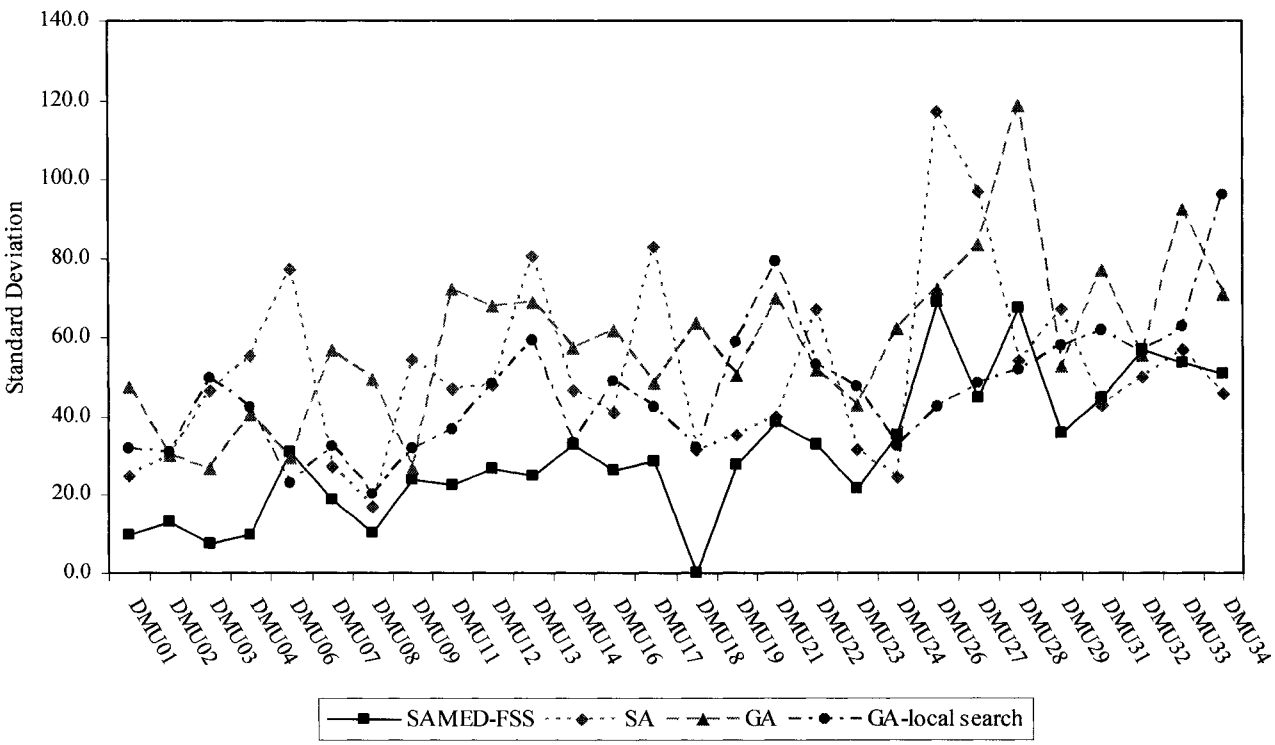
Table 6-7. Comparison of the standard deviations

Taillard's benchmark problems					Demirkol *et al.* benchmark problems				
	SAMED-FSS	SA[a]	GA[b]	GA-LS[c]		SAMED-FSS	SA[a]	GA[b]	GA-LS[c]
Ta01	8.9	9.2	8.8	0.0	DMU01	9.8	24.8	47.6	31.9
Ta02	0.4	1.7	8.2	5.8	DMU02	13.2	30.0	29.9	31.1
Ta03	3.2	5.7	19.0	15.6	DMU03	7.6	46.5	26.6	49.9
Ta04	2.8	4.9	20.7	19.6	DMU04	10.0	55.6	40.5	42.3
Ta05	3.5	3.0	28.4	8.7	DMU06	31.0	77.2	29.8	23.1
Ta11	6.2	19.7	15.6	16.5	DMU07	19.0	27.5	57.0	32.6
Ta12	9.1	17.1	26.1	24.7	DMU08	10.5	17.1	49.5	20.0
Ta13	5.6	21.1	19.8	11.6	DMU09	24.0	54.4	26.6	32.1
Ta15	5.6	22.7	29.9	23.1	DMU11	22.3	47.1	71.9	36.8
Ta21	7.2	14.1	44.9	22.4	DMU12	26.8	48.0	67.8	48.3
Ta24	9.2	27.7	27.3	15.5	DMU13	24.9	80.6	68.8	59.1
Ta25	8.5	24.6	21.5	14.4	DMU14	33.0	46.4	57.2	33.2
Ta31	1.6	2.4	17.3	9.2	DMU16	26.1	41.1	61.6	48.7
Ta33	2.3	7.3	9.7	14.9	DMU17	28.8	82.7	48.4	42.5
Ta34	4.5	9.6	23.9	23.6	DMU18	0.0	31.5	63.2	31.9
Ta35	0.5	5.1	14.2	14.6	DMU19	27.5	35.3	50.2	58.7
Ta41	15.3	24.4	20.6	44.2	DMU21	38.3	39.9	69.7	78.8
Ta42	19.6	17.1	57.8	27.6	DMU22	33.1	66.5	51.6	53.0
Ta44	15.7	15.6	55.3	32.6	DMU23	21.4	31.5	42.6	47.5
Ta48	10.5	13.8	46.0	25.9	DMU24	35.0	24.5	62.1	32.5
Ta61	3.2	9.8	22.5	15.2	DMU26	68.4	117.1	72.1	42.3
Ta65	1.8	17.7	23.2	24.5	DMU27	44.7	96.9	83.0	48.4
Ta66	4.5	4.0	19.7	16.7	DMU28	67.3	54.2	118.5	51.9
Ta69	12.0	13.1	28.2	12.2	DMU29	35.9	66.9	52.8	57.8
Ta71	14.9	13.6	51.3	37.9	DMU31	44.5	42.8	76.7	61.4
Ta72	9.1	7.3	17.7	47.5	DMU32	56.7	49.8	55.6	56.3
Ta73	23.8	25.1	32.5	19.0	DMU33	53.5	56.8	92.3	62.5
Ta76	6.8	12.0	20.7	32.2	DMU34	50.8	45.5	70.4	96.0

Note: a) Conventional SA; b) Standard GA; c) GA with local search



(a) Taillard's benchmark problems



(b) Demirkol *et al.* benchmark problems

Figure 6-10. Comparison of standard deviations

Table 6-8. Mean percentage deviation from Taillard's upper bound

Problem Size	GA Reeves (1995)	SA Reeves (1995)	ACS Ying &Liao (2004)	SAMED-FSS
20 × 5	1.61	1.27	1.19	0.00
20 × 10	2.29	1.71	1.7	0.11
20 × 20	1.95	0.86	1.6	0.56
50 × 5	0.45	0.78	0.43	0.03
50 × 10	2.28	1.98	1.89	0.37
100 × 5	0.23	0.56	0.22	0.06
100 × 10	1.25	1.33	1.22	0.12

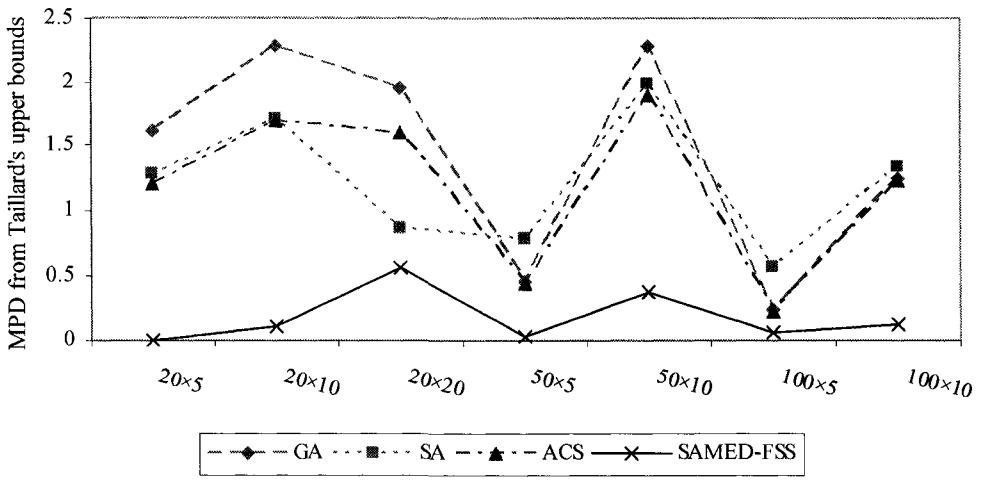


Figure 6-11. Comparison of mean percentage deviation from Taillard's upper bound (Pairwise exchange neighbourhood function)

6.4.2 The insertion technique

In the above experiments, both SAMED-FSS and the stand-alone SA algorithms employ the pairwise exchange scheme to generate a neighbourhood solution. In the pairwise neighbourhood generation technique, the positions of two randomly selected genes are swapped to generate a candidate solution. Alternately, another widely used neighbourhood generation scheme, the insertion technique, can also be adopted. To generate a neighbouring solution using this method, a gene on the chromosome of the current solution is selected randomly and inserted in front or back of another gene. Then in both SAMED algorithm and stand-alone SA, the pairwise neighbourhood generation scheme can be replaced with the insertion technique and the above experiment is repeated for all the methods and benchmark problems.

The parameters of the SAMED-FSS with insertion technique are presented in Table 6-9. The utilized values for the parameters such as initial temperature, control parameter, diversification coefficient, population size, probability of crossover, and probability of the mutation are the same as in the previous experiments. In the Taillard benchmark problems, the computational time (seconds) is limited to 5 for Ta02-Ta05; 180 for Ta11-Ta15; 120 for Ta21, Ta22, Ta24, and Ta25; 20 for Ta31- Ta35; 300 for Ta41, Ta42, Ta44, and Ta48; 90 for Ta61, Ta65, Ta66, and Ta69; and 600 for Ta71-Ta73 and Ta76. For the benchmark problems from Demirkol *et al.* (1998), the computational time (seconds) is 70 for DMU01-DMU04, 100 for DMU06-DMU09, DMU11-DMU14, and DMU16-DMU19; 120 for DMU21-DMU24; and 180 for DMU26-DMU29 and DMU31-DMU34.

Table 6-10 presents the computational results for the Taillard's benchmark problems. In this table the performance of the SAMED-FSS with two neighbourhood generation schemes, the pairwise exchange and insertion technique, is compared. According to the results presented in Table 6-10, the SAMED-FSS with insertion neighbourhood function finds more optimum solutions (20 out of 28) than the algorithm with pairwise exchange mechanism. Furthermore, for three test problems it finds solutions with shorter makespan than that of the other algorithm. Moreover, in the majority of cases, the computational times for the SAMED-FSS with insertion technique are significantly shorter that of the other algorithm with pairwise exchange neighbourhood function.

The computational results of the SAMED-FSS using the two neighbourhood generation methods and comparisons for the Demirkol *et al.* (1998) benchmark problems are presented in Table 6-11. Similar to that observed in the Taillard's benchmark problems, the SAMED-FSS with insertion neighbourhood technique clearly outperforms the algorithm with pairwise exchange neighbourhood generation scheme as it provides solutions with shorter makespan in 26 out of 28 cases.

Moreover, the results of the Mean Percentage Deviation (MPD) from Taillard's (mean) upper bounds for SAMED-FSS with insertion technique are summarized in Table 6-12 and compared to those reported in the literature (Ying and Liao, 2004) using several other heuristics. The comparison of the results has been also illustrated in Figure 6-12.

Table 6-9. Parameters of the SAMED-FSS with insertion neighbourhood function

	Problem size($n \times m$)	Short term memory (size)	Long term memory (size)	γ
Taillard's Benchmark problems	20 × 5	5	10	2
	20 × 10	10	30	2
	20 × 20	10	30	2
	50 × 5	10	30	0.5
	50 × 10	10	30	0.5
	100 × 5	10	30	0.25
	100 × 10	10	100	0.25
Demirkol *et al.* benchmark problems	20 × 15	10	30	2
	20 × 20	10	30	2
	30 × 15	10	30	2
	30 × 20	10	30	1
	40 × 15	10	30	1
	40 × 20	10	30	1
	50 × 15	10	30	1

Note: γ is the coefficient of the maximum number of unacceptable moves

Table 6-10. Comparison of the results for pairwise exchange and insertion neighbourhood functions (Taillard's problems)

Size ($n \times m$)	Problem	Lower bound	Optimal	Pairwise exchange neighbourhood				Insertion neighbourhood function			
				SA[a]	GA[b]	GA-LS[c]	SAMED-FSS[d]	SA[a]	GA[b]	GA-LS[c]	SAMED-FSS[d]
20 × 5	Ta01	1232	1278	1278(7)	1297(1)	1297(1)	**1278**(4)	1278(2)	1297(<1)	1278(1)	**1278(<1)**
	Ta02	1290	1359	1359(8)	1360(16)	1366(2)	**1359(1)**	1359(1)	1365(1)	1366(1)	**1359(<1)**
	Ta03	1073	1081	1081(14)	1107(8)	1098(2)	**1081(3)**	1083(5)	1108(1)	1091(2)	**1081(<1)**
	Ta04	1268	1293	1297(20)	1323(17)	1305(3)	**1293(9)**	1302(2)	1326(1)	1311(2)	**1293(3)**
	Ta05	1198	1235	1235(6)	1250(16)	1245(3)	**1235(2)**	1235(1)	1250(1)	1250(1)	**1235<1**
20 × 10	Ta11	1448	1582	1607(129)	1631(8)	1594(6)	**1582(187)**	1593(110)	1622(3)	1586(4)	**1582(41)**
	Ta12	1479	1659	1679(119)	1709(54)	1671(11)	**1659(134)**	1676(33)	1710(4)	1692(1)	**1659(67)**
	Ta13	1407	1496	1517(212)	1525(53)	1524(5)	**1496(225)**	1524(38)	1540(1)	1515(1)	**1496(156)**
	Ta15	1325	1419	1445(115)	1438(59)	1439(6)	**1419(239)**	1435(32)	1481(4)	1435(2)	**1419(95)**
20 × 20	Ta21	1911	2297	2332(295)	2329(35)	2331(29)	2302(214)	2320(38)	2325(2)	2315(7)	**2297(12)**
	Ta24	1810	2223	2243(13)	2271(43)	2246(29)	**2223(119)**	2254(7)	2264(5)	2246(29)	**2223(17)**
	Ta25	1899	2291	2323(12)	2364(35)	2309(46)	2294(186)	2315(83)	2325(6)	2309(46)	**2291(43)**
50 × 5	Ta31	2712	2724	2724(29)	2731(8)	2724(6)	**2724(8)**	2724(2)	2729(17)	2741(1)	**2724(1)**
	Ta33	2596	2621	2622(26)	2657(32)	2646(8)	**2621(22)**	2622(4)	2657(1)	2650(6)	**2621(3)**
	Ta34	2740	2751	2753(17)	2782(19)	2770(17)	**2751(23)**	2753(7)	2789(6)	2776(4)	**2751(7)**
	Ta35	2837	2863	2864(7)	2887(4)	2886(29)	**2863(10)**	2864(1)	2887(1)	2864(8)	**2863(6)**
50 × 10	Ta41	2907	2991	3063(169)	3140(69)	3094(51)	3030(261)	3059(208)	3116(71)	3111(21)	3035(148)
	Ta42	2821	2867	2928(298)	3008(141)	2971(64)	2911(47)	2921(197)	3006(77)	2975(9)	2911(97)
	Ta44	2968	3063	3079(289)	3147(158)	3113(44)	3067(171)	3071(298)	3194(71)	3121(17)	3066(217)
	Ta48	2959	3037	3054(177)	3153(87)	3097(50)	3046(199)	3048(230)	3121(88)	3108(10)	3044(116)
100 × 5	Ta61	5437	5493	5495(15)	5505(16)	5495(20)	**5493(39)**	5493(22)	5498(1)	5495(2)	**5493(9)**
	Ta65	5195	5250	5253(54)	5261(26)	5255(106)	**5250(78)**	5253(43)	5311(7)	5255(19)	**5250(83)**
	Ta66	5063	5135	5137(26)	5160(11)	5146(15)	**5135(62)**	5135(27)	5146(8)	5146(12)	**5135(21)**
	Ta69	5385	5448	5454(98)	5487(91)	5473(23)	**5448(87)**	5454(7)	5504(21)	5467(12)	**5448(35)**
100 × 10	Ta71	5759	5770	5785(497)	5887(82)	5843(294)	5774(770)	5783(490)	5950(163)	5855(43)	5773(508)
	Ta72	5345	5349	5384(627)	5459(762)	5406(240)	5362(674)	5377(171)	5494(156)	5425(44)	5362(284)
	Ta73	5623	5676	5691(494)	5777(887)	5738(327)	5679(298)	5687(144)	5818(117)	5738(34)	5679(196)
	Ta76	5246	5303	5321(643)	5391(817)	5349(142)	5308(683)	5321(122)	5406(137)	5344(42)	5308(67)

Note: a) Conventional SA; a) Standard GA; c) GA with Local Search; d) makespan (time/sec.), **bold**= optimal solution.

Table 6-11. Comparison of the results for pairwise exchange and insertion neighbourhood functions (Demirkol *et al.* problems)

Size ($n \times m$)	Problem	Demirkol et.al 1998	Pairwise exchange neighbourhood				Insertion neighbourhood			
			SA[a]	GA[b]	GA-LS[c]	SAMED-FSS[d]	SA[a]	GA[b]	GA-LS[c]	SAMED-FSS[d]
20 × 15	DMU01	4437(69)	3984 (34)	3981(10)	3954(19)	3933(135)	3965(5)	3981(55)	3981(5)	***3899*(4)**
	DMU02	4144(0.09)	3851(10)	3855(43)	3816(14)	3792(279)	3838(2)	3878(2)	3833(2)	***3761***(13)
	DMU03	3779(58)	3590(10)	3633(6)	3566(24)	3542(266)	3579(8)	3625(1)	3572(3)	***3534*(7)**
	DMU04	4302(67)	4074(63)	4175(11)	4088(12)	4058(106)	4097(6)	4148(25)	4094(4)	***4032(51)***
20 × 20	DMU06	4821(159)	4662(98)	4652(11)	4603(49)	4523(202)	4609(5)	4634(78)	4577(6)	***4523*(40)**
	DMU07	4779(148)	4552(78)	4552(49)	4479(46)	4459(44)	4544(12)	4561(9)	4482(10)	***4428*(23)**
	DMU08	4944(176)	4619(9)	4633(4)	4577(29)	4534(271)	4601(6)	4604(30)	4531(8)	***4523***(79)
	DMU09	4886(203)	4559(7)	4607(6)	4529(26)	4505(133)	4590(6)	4574(34)	4546(8)	***4496*(48)**
30 × 15	DMU11	5226(149)	4655(124)	4703(66)	4654(32)	4615(135)	4612(36)	4681(74)	4682(6)	***4582*(22)**
	DMU12	5304(163)	4775(107)	4812(124)	4751(60)	4709(150)	4742(55)	4730(79)	4725(10)	***4675*(61)**
	DMU13	5079(109)	4662(54)	4748(138)	4654(40)	4642(251)	4670(13)	4771(46)	4704(8)	***4569*(99)**
	DMU14	5605(0.17)	4906(209)	5016(88)	4929(36)	4877(295)	4894(40)	5014(38)	4949(12)	***4836*(54)**
30 × 20	DMU16	6183(0.24)	5492(48)	5537(171)	5466(85)	5431(170)	5431(48)	5494(59)	5494(19)	***5384*(76)**
	DMU17	6037(471)	5829(85)	5924(87)	5786(115)	5748(184)	5815(4)	5854(53)	5794(17)	***5718***(75)
	DMU18	6241(394)	5815(77)	5870(59)	5815(102)	5815(34)	5815(4)	5856(60)	5815(22)	***5726***(55)
	DMU19	6095(320)	5613(47)	5637(89)	5565(109)	5537(35)	5546(57)	5654(34)	5537(20)	***5479*(68)**
40 × 15	DMU21	6986(156)	6131(36)	6157(293)	6102(103)	6051(326)	6089(69)	6229(83)	6073(17)	***6026***(119)
	DMU22	6351(224)	5806(60)	5933(281)	5784(89)	5751(107)	5797(95)	5931(101)	5829(14)	***5717*(100)**
	DMU23	6506(289)	5964(137)	6024(67)	5959(97)	5929(86)	5944(86)	6105(60)	5997(15)	***5904*(40)**
	DMU24	6845(186)	5982(281)	6104(88)	5998(56)	5928(118)	5967(33)	6081(102)	6039(13)	5928(55)
40 × 20	DMU26	7154(615)	6628(245)	6777(217)	6716(159)	6603(261)	6711(31)	6771(98)	6763(34)	***6556*(176)**
	DMU27	7528(645)	6788(72)	6911(163)	6869(169)	6769(293)	6833(140)	6972(161)	6824(35)	***6704*(178)**
	DMU28	7469(674)	6945(50)	7086(299)	6933(241)	6888(285)	6965(173)	7033(160)	6962(29)	***6860*(180)**
	DMU29	7608(682)	6844(173)	6996(251)	6867(139)	6856(164)	6834(131)	7000(133)	6935(23)	***6792*(166)**
50 × 15	DMU31	7673(313)	6795(220)	6875(169)	6792(111)	6772(318)	6747(92)	6952(76)	6852(14)	6747(80)
	DMU32	7679(299)	6724(162)	6857(299)	6750(171)	***6647***(406)	6740(172)	6844(118)	6761(28)	6669**(89)**
	DMU33	7416(284)	6736(205)	6834(295)	6780(101)	6658(241)	6733(134)	6893(144)	6748(31)	***6656*(138)**
	DMU34	7548(307)	6945(223)	7054(417)	6940(144)	6900(315)	6945(149	7125(159)	6972(28)	***6878*(101)**

Note: a) Conventional SA; b) Standard GA; c) GA with Local Search; d) makespan (time/sec.), ***italic bold***= best solution

Table 6-12. Mean percentage deviation from Taillard's upper bound

Problem Size	GA Reeves (1995)	SA Reeves (1995)	ACS Ying &Liao (2004)	SAMED-FSS with the insertion neighbourhood generation method
20 × 5	1.61	1.27	1.19	0.00
20 × 10	2.29	1.71	1.7	0.00
20 × 20	1.95	0.86	1.6	0.03
50 × 5	0.45	0.78	0.43	0.00
50 × 10	2.28	1.98	1.89	0.03
100 × 5	0.23	0.56	0.22	0.00
100 × 10	1.25	1.33	1.22	0.02

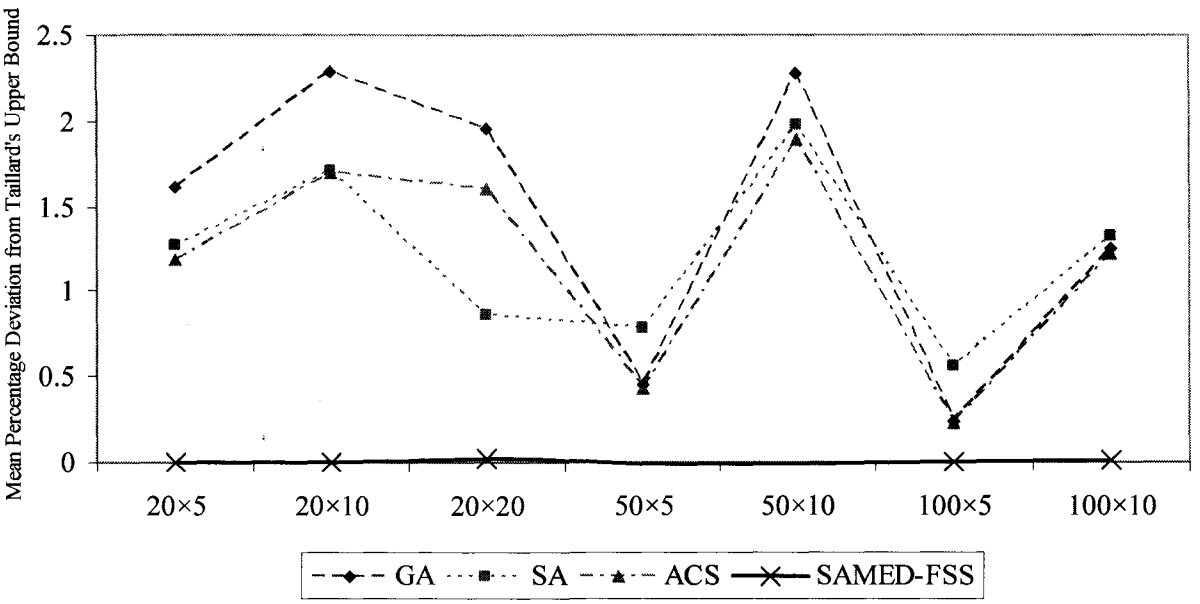


Figure 6-12. Comparison of mean percentage deviation from Taillard's upper bound (Insertion neighbourhood function)

Chapter 7

Conclusion and Future Work Directions

7.1 Conclusion

This thesis focuses mainly on four topics: job boredom measuring and prediction, job rotation methodology and related implementation issues to reduce employee's boredom in a manufacturing system, metaheuristic search technique to solve the job rotation problem, and machine scheduling. Chapter 3 introduces a probabilistic framework to measure and predict boredom at work. In chapter 4, a methodology to incorporate both employee's boredom and skill variations in designing rotation plans is presented. In chapter 5, first a generic framework for a new metaheuristic (SAMED) is proposed. Then based on the proposed framework, a search algorithm is tailored to solve the job rotation problem presented in chapter 4. The application of the proposed algorithm to two well known scheduling problems is also investigated and presented in chapter 6. For this reason, chapter 6 is divided into two parts. Part 1 presents the application of the new technique to the job shop scheduling problem. The detailed discussion for the application of the SAMED to the flow shop scheduling problem is presented in part 2.

1. Modeling job boredom using Bayesian Networks

According to literature, job boredom has been commonly measured by self reporting scales. In this research, a novel approach based on the state-of-the-art Bayesian Networks has been explored to model employee's boredom at work. The proposed probabilistic model could be used to measure and predict job boredom quantitatively. A static Bayesian Networks is first presented to capture the static aspect of boredom modeling. To account for temporal aspect of boredom modeling, a dynamic Bayesian Networks is also presented. The relevant variables to job boredom as well as the relationships between

those variables are identified using expert knowledge (i.e., literature). Due to the lack of available data, the quantitative part of the model (conditional probability tables) is specified using two approaches, a combination of qualitative explanation approach and a weighting scheme and noisy-OR gate principle. The implementation and inferences are carried out using GeNIe environment (http://genie.sis.pitt.edu/). To validate the proposed model, several scenarios representing real life situations are generated. The computational results for target hypothesis (i.e., boredom) may, to some extent, confirm the validity of the model.

2. Modeling job rotation in a manufacturing system

In this case, workers are assigned to different workstations. A methodology is presented to incorporate both employee's boredom and skill variations in job rotation schemes. According to the presented methodology, worker skill variations are formulated by generalizing the concepts of workers learning and forgetting phenomena. The variation in worker's boredom is assumed to follow an exponential pattern. Based on the proposed formulations for boredom and skill variations, a mathematical programming model is developed for job rotation in a manufacturing cell. The proposed model is categorized as a nonlinear mixed integer mathematical programming model. A linear variant of the model in which the change in workers boredom and skill are assumed linear is also presented. Both models are examined using numerical examples.

3. A new metaheuristic approach for combinatorial optimization problem

A generic framework of a new metaheuristic based on several well known heuristic search algorithms: simulated annealing, tabu search, and genetic algorithm is proposed to solve combinatorial optimization problems in general and the job rotation/scheduling problems in particular. The performance of the proposed metaheuristic (SAMED) has thus far been evaluated using two NP-hard scheduling problems, job shop and flow shop. Detailed discussions on the computational results pertaining to each experiment are presented as follows:

3.1 Job rotation problem

Based on the generic framework of a metaheuristic, a search algorithm, SAMED-JR is developed to solve the proposed job rotation model. To compare the performance of the SAMED-JR with other algorithms, the two popular heuristics, genetic algorithm and simulated annealing, have been also developed. The three algorithms have been applied to 16 randomly generated job rotation problems. The computational results pertaining to the benchmark problems clearly indicate the superior performance of the SAMED-JR over the other two algorithms both in terms of the solution's quality and the speed of convergence.

3.2 Job shop scheduling

The SAMED-JSS algorithm includes a SA module, three types of memories, and a GA based evolutionary operator. This algorithm as a metaheuristic can be "customized" for enhanced performance in specific applications. In this study, this is demonstrated by augmenting it with a local search module and a blockage removal feature specially tailored for the job shop scheduling problem. Both the SAMED-JSS and its augmented version, SAMED-LB, have been tested using classical job shop scheduling problems. The SAMED-JSS algorithm finds the optimal solution for 29 instances (78.4%) of the 37 problems from the literature within reasonably short computational times. For the remaining problems, near optimal solutions are obtained.

While the SAMED-JSS has shown superior performance as a general purpose search method, the computational comparison with SAMED-LB confirms that the new feature i.e., local search, can further improve the performance in solving the scheduling problems. The SAMED-LB finds the optimal solution for nearly 84% of the test problems. The results also show that the SAMED-LB can locate the optimal solutions for medium and large size problems in much shorter times (about 16 to 90 % faster than the SAMED-JSS). The performance of the proposed methods has been compared to several well-known algorithms including exact methods, general techniques (e.g., GA, SA, TS and GRASP) and hybrid algorithms. The results clearly indicate the superior performance of the proposed metaheuristic over general and hybrid algorithms in terms of both solution quality and computational efficiency.

3.3 Flow shop scheduling

A search algorithm based on the generic framework of the SAMED, SAMED-FSS, is developed to solve the flow shop scheduling problem. To examine the performance of the SAMED-FSS algorithm, codes also developed for the stand-alone SA and GA, as well as a GA with a local search feature (GA-LS).

The performance of the developed algorithms on the permutation flow shop scheduling is evaluated using two sets of hard benchmark problems from Taillard (1990) and Demirkol *et al.* (1998). Our computational results indicate that the stand-alone GA exhibits relatively poor performance in solving the flow shop scheduling problem. In general, the stand-alone SA performs better than GA-LS in Taillard's benchmark problems. However, the same conclusion could not be drawn from the computational results pertaining to the Demirkol *et al* test problems. For most of these problems, the GA-LS obtained better solutions than those by the stand-alone SA.

Among the four algorithms, i.e., stand-alone SA, stand-alone GA, GA-LS and SAMED-FSS, SAMED-FSS consistently outperforms the other three. It also compares favourably with the PSO method in both benchmark problem sets. The standard deviations of the makspans over several runs for each test problem and algorithm were also evaluated and compared. The results show that SAMED-FSS is not only capable of providing good solutions, it also provides solutions with significantly lower standard deviations in the majority of the cases that have been investigated. This is an indication of better robustness or consistency.

The solutions obtained by SAMED-FSS for the Taillard's benchmark problems were also compared favourably with those provided by the two well-known tabu search methods (Taillard 1990, Ben-Daya and Al-fawzan 1998).

The effect of neighbourhood function on the performance of the proposed method has been also investigated. The computational results for both benchmark problem sets show that in the majority of cases the algorithm with insertion technique finds better or the same solutions in shorter computational time.

7.2 Future work

The following works may require further investigation.

1. Job rotation model

Studies have shown that firms with flexible workers deal with uncertainties better than those with non-flexible workers. However, training multi skills worker is costly and required continues attention. Investigating job rotation methodologies that incorporate worker's skill variations and manufacturing flexibility would be an interesting subject to explore. Here the manufacturing flexibility is defined as the capability of a manufacturing system to deal with changes thorough multi skills workers.

2. Extension of the Bayesian model

The proposed Bayesian model to measure and predict job boredom has been tested using several scenarios. However, further research is required to investigate the efficiency of the model using data from real world. Furthermore, for simplicity, the variables in the boredom model are assumed to take only two states (i.e., binary variables). The proposed model can be further extended by increasing the number of states that variables could take. This may increase the accuracy of the model outcomes (i.e., inferences). Finally, further study is necessary to determine whether or not there exist other variables that may also contribute in employee's boredom feeling and/or be regarded as boredom effects.

3. New diversification component

The proposed metaheuristic utilizes a population list and an evolutionary operator to diversify the search in due course. The evolutionary operator in the framework could be replaced by a Particle Swarm Optimization (PSO) module. The PSO module may improve the diversification phase of the algorithm which is an essential part of the proposed metaheuristic.

4. Solution to the job rotation problem

Solutions to the job rotation problem are represented by chromosomes. The chromosomes are then converted to matrices to reduce computational burden. However, subjecting solutions with such representation scheme to minor or major perturbations may result in producing unfeasible solutions. This shortcoming may reduce the performance of the proposed solution technique, i.e., the SAMED-JR algorithm. As a result, further research may be necessary to design a solution representation for the problem that guarantees the feasibility of the solutions produced by a neighbourhood generator function and/or a crossover operator.

Furthermore, to solve the job rotation problems, the initial solutions have been generated randomly. Studies show that starting the search from a good initial solution reduces the computational times as well as the quality of the final solutions. Therefore, it is suggested to investigate the possibility of designing a problem-specific heuristic to generate good initial solutions for the proposed algorithm.

Bibliography

Aarts, E.H.L., van Laarhoven, P.J.M, Lenstra, J.K., and Ulder, N.L.J., 1994, "A computational study of local search algorithms for job shop scheduling," *ORSA Journal on Computing*, **6**, pp.118-125.

Adams, J., Balas, E., and D., Zawack, 1988, "The shifting bottleneck procedure for job shop scheduling," *Management Science*, **34**, pp. 391-401.

Ahmed, S.M., 1990, "Psychometric properties of the boredom proneness scale," *Perceptual and Motor Skills*, **71**, pp. 963-6.

Aiex, R.M., Binato, S., Resende, M.G.C., 2003, "Parallel GRASP with path relinking for job shop scheduling," *Parallel Computing*, **29**, pp. 393-430.

Anselmi, F., Sundarrajan, S., 2000 "Aligning workers and workloads," IIE Sol., pp.28-32.

Applegate, D., Cook, W., 1991, "A computational study of job shop scheduling problem," *ORSA Journal on Computing*, **3**, pp.149-156.

Ashkanasy, N.M., Hartel, C.E.J., and Daus, C.S., 2002, "Diversity and emotion: the new frontiers in organizational behavior and research," *Journal of Management*, **28**, pp.307-38.

Ashour, S., Hiremath, S.R., 1973, "A branch-and-bound approach to the job-shop scheduling problem," *International Journal of Production Research*, **11**, pp. 47-58.

Askin, R. G., Huang, Y., 1997, "Employee training and assignment for facility reconfiguration," *in Proc. of 6th International Engineering Research Conference*, IIE, pp. 426-431.

Azizi, N., Zolfaghari, S., 2004, "Adaptive temperature control for simulated annealing: a comparative study," *Computers and Operations Research*, **31**, pp.2439-2451.

Bailey, C.D., 1989, "Forgetting and learning curve: A laboratory study," *Management Science*, **35**, pp.340-352.

Balas, E., Lenstra , J.K., and Vazacopoulos, A.,1995, "The one-machine problem with delayed precedence constraints and its use in job shop scheduling," *Management Science*, **41**, pp. 94-109.

Ben-Daya, M., Al-Fawzan, M., 1998, "A tabu search approach for the flow shop scheduling problem," *European Journal of Operational Research*, **109**, pp.88-95.

Bhadury, J., Radovilsky, Z., 2006, "Job rotation using the multi-period assignment model," *International Journal of Production Research*, **44**, pp.4431-4444.

Bhaskar, K., & Srinivasan, G., 1997, "Static and dynamic operator allocation problems in cellular manufacturing systems," *International Journal of Production Research*, **35**(12), pp. 3467-3481.

Binato, S.,, Hery, W.J., Loewenstern, D., and Resende, M.G.C., 2001, "A GRASP for job shop scheduling" Kluwer Academic Publishers, pp. 59-79.

Bierwirth, C., Mattfeld, D.C., Kopfer, H., 1996, "On Permutation Representations for Scheduling Problems," *in Proc. 4th International Conference on Parallel Problem Solving from Nature*, Lecture notes in computer science, Springer-Verlag, pp. 310–318.

Boese, K.D., Kahng, A.B., and Muddu, S., 1994, "A new adaptive multi-start technique for combinatorial global optimizations," *Operations Research Letters*, **16**, pp.101-113.

Bond, C. F., Jr., and Titus, L. J., 1983, "Social facilitation: A meta-analysis of 241 studies," *Psychological Bulletin*, **94**, pp.265-292.

Brown, S., Heathcote, A., 2003, "Averaging learning curves across and within participants," *Behavior Research Methods, Instruments, & Computers*, **35**, pp.11–21.

Campbell, H. G., Dudek, R. A., and Smith, M. L., 1970, "A heuristic algorithm for the n-job, m-machine sequencing problem," *Management Science*, **16**, pp.630-637.

Carlier, J., Pinson, E.,1989 "An algorithm for solving the job shop problem," *Management Science*, **35**, pp. 164-176.

Carlier, J., and Rebaï, I., 1996, "Two branch and bound algorithms for the permutation flow shop problem," *European Journal of Operational Research*, **90**, pp.238-251.

Carnahan, B.J., Redfern, M.S. and Norman B.A., 2000, "Designing safe job rotation schedules using optimization and heuristic search," Ergonomics, **43**, pp.543–560.

Cesani, V. I., Steudel, H. J., 2000b, "A model to quantitatively describe labour assignment flexibility in labour limited cellular manufacturing systems," Group

technology/cellular manufacturing world symposium, San Juan, Puerto Rico, pp. 159-164.

Chen, C.L., Vempati, V.S. and Aljaber, N., 1995, "An application of genetic algorithms for flow shop problems," *European Journal of Operational Research*, **80**, pp.389-396.

Cheng, R., 1997, "A study on genetic algorithms-based optimal scheduling techniques," PhD thesis, Tokyo Institute of Technology.

Colorni, A., Dorigo , M., and Maniezzo, V., 1994, "Ant system for job-shop scheduling," *Belgian Journal of Operations Research*, Statistics, and Computer Science, **34**, pp. 39-53.

Cunningham, B.J., Eberle, T., 1990, "A guide to job enrichment and redesign," *Personnel*, pp.56–61.

Daniel, BK., Zapata-Rivera, JD., Schwier, RA., and McCalla, GI., 2006, "Bayesian Belief Network Models of Trust and Social Capital for Social Software Systems Design," CHI, 'Reinventing trust, collaboration and compliance in social systems' workshop, Montréal, Québec, Canada.

Davis, L.E. and Taylor, J.C., 1989, "Design of Jobs, Goodyear Publishing: Santa Monica, CA.

Della Croce, F., Tadei, R., and Volta, G., 1995, "A genetic algorithm for the job shop problem," *Computers & Operations Research*, **22**, pp. 15-24.

Dell'Amico, M., Trubian, M., 1993, "Applying tabu search to the job shop scheduling problem," *Annals of Operations Research*, **41**, pp. 231-252.

Demirkol, E., Mehta, S., and Uzsoy, R., 1998, "Benchmarks for shop scheduling problems," *European Journal of Operational Research*, **109**, pp.137-141.

Díez, FJ., 1993, "Parameter adjustment in Bayes networks: The generalized Noisy-OR gate,", *in Proceedings of the Ninth Annual Conference on Uncertainty in Artificial Intelligence* (UAI–93), pp. 99–105, Washington, D.C.

Díez, FJ., Druzdzel, M J., 2006, "Canonical probabilistic models for knowledge engineering," Technical Report CISIAD-06-01. UNED, Madrid.

Dorigo, M., Gambardella, L.M., 1997 "Ant colony system: A cooperative learning approach to the traveling salesman problem," *IEEE Transaction on Evolutionary Computation*, pp.53-66.

Drory, A., 1982, "Individual differences in boredom proneness and task effectiveness at work," *Personal Psychology*, **35**, pp.141-151.

Druzdzel, M., Henrion, M., 1993b, "Efficient reasoning in qualitative probabilistic networks," *in Proceedings of the 11th National Conference on Artificial Intelligence*, pp.548-553.

El-Bouri, A., Azizi, N., Zolfaghari, S., 2007, "A comparative study of new metaheuristics based on simulated annealing and adaptive memory programming," *European Journal of Operations Research*, **177**, pp.1894-1910.

Farmer, R., and Sundberg, N. D., 1986, "Boredom proneness-The development and correlates of a new scale," *Journal of Personality Assessment*, **50**, pp.4-17.

Fenton, N. E., Neil, M., and Caballero, J.G., 2007, "Using Ranked Nodes to Model Qualitative Judgments in Bayesian Networks," *IEEE Transaction on Knowledge and Data Engineering*, **19**(10), pp.1420-1432.

Fisher, C. D., 1993, "Boredom at work: A neglected concept," *Human Relations*, **46**, pp.395–417.

Fisher, C.D., 1987, "Boredom: Construct, Causes and Consequences," A&M University, Texas, Technical Report ONR-9.

Fisher, H., Thompson, G.L., 1963, "Probabilistic learning combinations of local job-shop scheduling rules," In: Muth, J.F., Thompson, G.L. (Eds.), Industrial Scheduling. Prentice- Hall, Englewood Cliffs, NJ, pp. 225-251.

Feo, T., Resende, M., 1995, "Greedy randomized adaptive search procedures," *Journal of Global Optimization*, **16**, pp. 109-133.

Framinan, J.M., Gupta, J.N.D., and Leisten, R., 2004, "A review and classification of heuristics for permutation flow shop scheduling with makespan objective," *Journal of Operational Research Society*, **55**, pp.1243-1255.

Gambardella, L.M., Dorigo, M., 1996, "Solving symmetric and asymmetric TSP's by ant colonies," *in Proc. IEEE Conference on Evolutionary Computation* (ICEC'96), pp. 622-627.

Game, A.M., 2007, "Workplace boredom coping: health, safety, and HR implications," *Personnel Review*, **36**(5), pp.701-721.

Gardner, D. G., Cummings, L. L., 1988, "Activation theory and job design: review and re-conceptualization," *Research in Organizational Behaviour*, **10**, pp.81-122.

Garey, M.R., Johnson, D.S., and Sethi, R., 1976, "The complexity of flow shop and job shop scheduling," *Mathematics of Operations Research*, **1**, pp.117-129.

Globerson, S., Levin, N., and Shtub, A., 1989, "The impact of breaks on forgetting when performing a repetitive task," *IIE Transactions*, **21**, pp.376-381.

Glover, F., 1986, "Future paths for integer programming and links to artificial intelligence," *Computers and Operations Research*, **13**, pp.533-549.

Gonçalves, J.F., Mendes, J.J.M., Resende, M.G.C., 2005, "A hybrid genetic algorithm for the job shop scheduling problem," *European Journal of Operational Research*, **167**, pp. 77-95.

Grabowski, J., Wodecki, M., 2004, "A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion," *Computers and Operations Research*, **31**, pp.1891–1909.

Grubb, E. A., 1975, "Assembly line boredom and individual differences in recreation participation," *Journal of Leisure Research*, **7**, pp.256-269.

Gupta, J.N.D, 1971a, "Flow shop scheduling via sorting analogy," UARI Research Report No. 109, University of Alabama.

Gupta, J.N.D., 1971b, "A functional heuristic for the flow-shop scheduling problem," *Operational Research Quarterly*, **22**, pp.39-47.

Guest, D., Williams, R., and Dewe, P., 1978, "Job design and the psychology of boredom," *in Prec. of 19th Congress of Applied Psychology*, Munich, West Germany.

Held, M., and Karp, R. M., 1962, "A dynamic programming approach to sequencing problems," *Journal of SIAM*, **10**, pp.196-210.

Henderson, C.J., 1992, "Ergonomic job rotation in poultry processing," *Advances in Industrial Ergonomics and Safety*, **4**, 443–450.

Henrion, M., 1989, "Some practical issues in constructing belief networks," in Uncertainty in Artificial Intelligence 3, eds., L.N. Kanal, T.S. Levitt, and J.F. Lemmer, , Elsevier Science Publishers B.V., North Holland, pp.161–173.

Heragu, S. S., 1994, "Group technology and cellular manufacturing," *IEEE Transactions of Systems, Man, and Cybernetics*, **24**(2), pp.203-215.

Hill A. B., Perkins R. E., 1985, "Towards a model of boredom," *British journal of psychology*, **76**(2), pp. 235-240.

Hinnen, U., Laubli, T., Guggenbuhl, U., and Krueger, H., 1992, "Design of check-out systems including laser scanners for sitting work posture," *Scandinavian Journal of Work, Environment and Health*, **18**, pp.186–194.

Holland, J.H., 1975, "Adaptation in Natural and Artificial Systems," The University of Michgan Press, Ann Arbor, MI.

Ishibuchi, M., Misaki, S. and Tanaka, H., 1995, "Modified simulated annealing for the flow shop sequencing problems," *European Journal of Operational Research*, **81**, pp.388-398.

Jaber, M.Y., Bonney, M., 1996, "Production breaks and the learning curve: The forgetting phenomenon," Applied Mathematical Modelling, 20, pp. 162–169.

Jaber, M.Y., Kher, H.V., 2004, "Variant versus invariant time to total forgetting: The learn–forget curve model revisited," Computer and Industrial Engineering, 46, pp. 697–705.

Jain, A.S., Meeran, S., 1999, "Deterministic job-shop scheduling: Past, present and future," *European Journal of Operational Research*, **113**, pp. 390-434.

Ji, Q., Lan, P., and Looney,C., 2006, "A Probabilistic Framework for Modeling and Real-Time Monitoring Human Fatigue," *IEEE Transactions on Systems, Man, and Cybernetics*, part A: systems and humans, **36**(5), pp.862-875.

Johnson, S.M., 1954, "Optimal two- and three-stage production schedules with set-up times included," *Naval Research, Logistic Quarterly*, **1**, pp.61–68.

Lageweg, B.J., Lenstra, J.K., and Rinnooy Kan, A.H.G., 1978, "A general bounding scheme for the permutation flow shop problem," *Operations Research*, **26**, pp.53-67.

Lawrence, S., 1984. Supplement to resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques. Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA.

Kennedy, J., Eberhart, R., 1995, "Particle swarm optimization," *in Proc. IEEE International Conference on Neural Network*, pp.1942-1948.

Kher H.V., Malhotra M.K.1, Philipoom P.R., and Fry T.D., 1999, "Modeling simultaneous worker learning and forgetting in dual resource constrained systems," *European Journal of Operational Research*, **115**, pp.158-172.

Kher, H. V., 2000, "Examination of flexibility acquisition policies in dual resource constrained job shops with simultaneous worker learning and forgetting effect," *Journal of Operational Research Society*, **51**, pp.592-601.

Kirkpatrick, S., Gelatt, C.D., and Vecchi, M.P., 1983, "Optimization by simulated annealing," *Science*, **220**, pp. 671-680.

Kolonko, M., 1999, "Some new results on simulated annealing applied to the job shop scheduling problem," *European Journal of Operations Research*, **113**, pp.123-136.

Lee, T. W., 1986, "Toward the development and validation of a measure of job boredom," *Manhattan College Journal of Business*, **15**, pp.22-28.

Liao, C.J., Tseng,C.T., Luarn, P., 2007, "A discrete version of particle swarm optimization for flow shop scheduling problems," *Computers & Operations Research*, **34**, pp.3099-3111.

Locke E.A., Latham G.P., 1990, "A Theory of Goal Setting and Task Performance," Englewood Cliffs, NJ: Prentice-Hall.

Lourenço, H.R., 1996, "Sevast'yanov's algorithm for the flow-shop scheduling problem," *European Journal of Operational Research*, **91**, pp.176-189.

MacDonald, S., MacIntyre, P., 1997, "The generic job satisfaction scale: Scale development and its correlates," *Employee Assistance Quarterly*, **13**, pp.1–16.

Mason, S.J., Fowler, J.W., and Carlyle, W.M., 2002 "A modified shifting bottleneck heuristic for minimizing total weighted tardiness in complex job shops," *Journal of Scheduling*, **5**, pp. 247-262.

Mattfeld, D. C., Bierwirth, C., 2004, "An efficient genetic algorithm for job shop scheduling with tardiness objectives," *European Journal of Operational Research*, **155**, pp. 616 -630.

Mihaljlovic, V., Petkovi, M., 2001, Dynamic Bayesian Networks: A state of the art, University of Twente, Enschede, The Netherlands, Available: http://doc.utwente.nl/fid/1228.

Mikulas, W. L., Vodanovich, S. J., 1993, "The essence of boredom, *The Psychological Record,* **43**, pp. 3–12.

Molleman, E., Slomp, J., 1999, "Functional flexibility and team performance," *International Journal of Production Research*, **37**(8), pp.1837-1858.

Murata, T., Ishibuchi, H., and Tanaka, H., 1996, "Genetic algorithms for flow shop scheduling problems," *Computers and Industrial Engineering*, **30**, 1061-1071.

Murata, T., Ishibuchi, H. and Tanaka, H., 1996, "Multi-objective genetic algorithm and its applications to flowshop-scheduling," *Computers & Industrial Engineering*, **30**, 957-968.

Nadkarnia, S., Shenoyb, P. P., 2004, "A causal mapping approach to constructing Bayesian networks," *Decision Support Systems*, **38**, pp.259- 281.

Nawaz, M., Enscore, E. E., and Ham, I., 1983, "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem," *OMEGA*, **11**, pp.91-95.

Norman, B. A., Tharmmaphornphilas, W., Needy, K. L., Bidanda, B., and Warner, R. C. ,2002, "Worker assignment in cellular manufacturing considering technical and human skills," *International Journal of Production Research*, **40**(6), pp.1479-1492.

Nowicki, E., Smutnicki, C., 1996, "A fast tabu search algorithm for the job shop problem," *Management Science*, **42**, pp. 797-813.

Nowicki, E., Smutnicki, C., 1996, "A fast tabu search algorithm for the permutation flow-shop problem," *European Journal of Operational Research*, **91**, pp.160-175.

Ogbu, F.A., and Smith, D.K., 1990, "The application of the simulated annealing algorithm to the solution of the n/m/Cmax flowshop problem," *Computers & Operations Research*, **17**, pp.243-253.

Osman, I.H. and Potts, C.N., 1989, "Simulated Annealing for Permutation Flow-shop Scheduling," *OMEGA*, **17**, pp.551-557.

Osman, I.H., 1993, "Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem," *Annals of Operations Research*, **41**, pp.421-451.

Palmer, D. S., 1965, "Sequencing jobs through a multistage process in the minimum total time: a quick method of obtaining a near-optimum," *Operational Research Quarterly*, **16**, pp.101-107.

Pearl, J., 1988, "Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference," San Mateo, CA: Morgan Kaufmann Publishers, Inc.

Prabhaharan, G., Shahul Hamid Khan, B., and Rakesh, L., 2006, "Implementation of grasp in flow shop scheduling," *International Journal of Advanced Manufacturing Technology*, **30**, pp.1126–1131.

Ragheb, M.G., Merydith, S.P., 2001, "Development and validation of a multidimensional scale measuring free time boredom," *Leisure Studies*, **20**, pp.41–59.

Reeves, C. R., 1995, "A genetic algorithm for flow shop sequencing," *Computers & Operations Research*, **22**, pp.5-13.

Reeves, C. Yamada, T., 1998, "Genetic algorithms, path relinking, and the flowshop sequencing problem," *Evolutionary Computation journal* (MIT press), **6**, pp.45-60.

Russell, S., Norvig, P., 2003, "Artificial Intelligence: A Modern Approach," Second Edition, Prentice Hall.

Russell, R.S., Taylor, B.W., 1998, Operations Management: focusing on quality and competitiveness, prentice Hall, Inc, second edition, pp.350-354.

Ryan, R. M., 1982, "Control and information in the intrapersonal sphere: An extension of cognitive evaluation theory," *Journal of Personality and Social Psychology*, **43**, pp.450-461.

Sawin, D. A., Scerbo, M. W., 1995, "Effects of instruction type and boredom proneness in vigilance: Implications for boredom and workload," *Human Factors*, **37**, pp.752-765.

Shafer, S.M., Nembhard, D.A., and Uzumeri, M.V., 2001, "The effects of worker learning, forgetting, and heterogeneity on assembly line productivity," *Management Science*, **47**, pp.1639-1653.

Smith, A.C., Ellesworth, P.C., 1985, "Pattern of cognitive appraisal in emotion," *Journal of Personality Social Psychology*, **48**, pp.813-838.

Stützle, T., 1998, "An ant approach for the flow shop problem," EUFIT, Aachen, Germany, pp.1560-1564.

Stutzle, T., Hoos, H., 2000, "MAX–MIN ant system," *Future Generation Computer Systems* (FGCS), **16**, pp. 889–914.

Taillard, E., 1990, "Some efficient heuristic methods for the flow-shop sequencing problem," *European Journal of Operational Research*, **47**, pp.65-74.

Taillard, E., 1993, "Benchmark for basic scheduling problems," *European Journal of Operational Research*, **64**, pp.278-285.

Taillard, E.D., Gambardella, L.M., Gendreau, M., Potvin, J.Y., 2001, "Adaptive memory programming: A unified view of metaheuristics," *European Journal of Operational Research*, **135**, pp. 1-16.

T'kindt, V., Monmarche, N., Tercinet, F., Laugt, D., 2002, "An ant colony optimization algorithm to solve a 2-machine bicriteria flow shop scheduling problem," *European Journal of Operational Research*, **142**, pp. 250–257.

Tharmmaphornphilas, W., Green, B., Carnahan, B.J. and Norman, B.A.,2003, "Applying mathematical modeling to create job rotation schedules for minimizing occupational noise exposure," *AIHA Journal*, 2003, **64**, pp.401–405.

Thomas, F., Stoner, J., and Warren, E. K., 1977, "Managing the career plateau," *Academy of Management Review*, **2**, pp.602-12.

Van Laarhoven, P.J.M., Aarts, E.H.L., Lenstra, J.K., 1992, "Job shop scheduling by simulated annealing," *Operation Research*, **40**, pp.113-126.

Wright, T., 1936, "Factors affecting the cost of airplanes," *Journal Aeronaut Science*, **3**, pp.122-128.

Vodanovich, S. J., 2003, "Psychometric measures of boredom: A review of the literature," *The Journal of Psychology*, **137**, pp.569–595.

Wang, L., Zheng, D.Z., 2001 "An effective hybrid optimization strategy for job-shop scheduling problems," *Computers & Operations Research*, **28**, pp. 585-596.

Wang, L., Zheng, D.Z., 2003, "An effective hybrid heuristic for flow shop scheduling," *International Journal of Advanced Manufacturing Technology*, **21**, pp.38-44.

Warner, R. C., Needy, K. L., and Bidanda, B., 1997, "Worker assignment in implementing manufacturing cells," 6th industrial engineering research conference proceedings, Miami Beach, FL, pp. 240-245.

Wellman, M.P., 1990, "Fundamental Concepts of Qualitative Probabilistic Networks," *Artificial Intelligence*, **44**(3), pp. 257-303.

William, A. Y., Dale, T. M., and Robert P. J., 2008, "A matrix-based methodology for determining a part family's learning rate," *Computers and Industrial Engineering*, **54**, pp. 390-400.

Xia, W., Wu, Z., 2006 "A hybrid particle swarm optimization approach for the job-shop scheduling Problem," *International Journal of Advanced Technology*, **29**, pp.360-366.

Yamada, T., Nakano, R., 1997, "Genetic algorithms for job-shop scheduling problems," *in Proc. Modern Heuristic for Decision Support*, UNICOM seminar, pp.67-81.

Ying, K. C., Liao, C. J., 2004, "An ant colony system for permutation flow-shop sequencing," *Computers & Operations Research*, **31**, pp.791-801.

Zagorecki, A., Druzdzel, M., 2004, "An Empirical Study of Probability Elicitation under Noisy-OR Assumption," *In Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference* (FLAIRS, 2004), Valerie Barr & Zdrawko Markov (eds), pp. 880-885.

Zuckerman, M., 1971, "Dimensions of sensation seeking," *Journal of Consulting and Clinical Psychology*, **36**, pp. 45-52.

Zuckerman, M., 1979, "Sensation Seeking: Beyond the Optimal Level of Arousal," Erlbaum, Hillsdale, NJ.

Zolfaghari, S., Liang, M., 1999, "Jointly solving the group scheduling and machining speed selection problems: A hybrid tabu search and simulated annealing approach," *International Journal of Production Research*, **37**, pp.2377–2397.

Appendices

Appendix A

Lee's Job Boredom Scale (Lee, 1986)

1	Do you often get bored with your work?	Yes	No
2	Is your work monotonous	Yes	No
3	Would you like to change from your type of work to another from time to time (if the pay were the same)?	Yes	No
4	How well do you like the work you do?	Yes	No
5	Do you often get tired on the job?	Yes	No
6	Do you find the job dull?	Yes	No
7	Does the job go by slowly?	Yes	No
8	Do you become irritable on the job?	Yes	No
9	Do you get apathetic on the job?	Yes	No
10	Do you get mentally sluggish during the day?	Yes	No
11	Do you get drowsy on the job?	Yes	No
12	Does the time seem to go by slowly?	Yes	No
13	Are there long periods of boredom on the job?	Yes	No
14	Does the job seem repetitive?	Yes	No
15	During the day, do you think about doing another task?	Yes	No
16	Does monotony describe your job?	Yes	No
17	Is your work pretty much the same day after day?	Yes	No

Appendix B

Job Rotation Benchmark Problems: Input Data

Parameters of JR-1 to JR-4

Worker	β	γ	τ	θ
1	-0.17	-0.18	0.11	0.10
2	-0.22	-0.09	0.15	0.09
3	-0.20	-0.12	0.17	0.08
4	-0.18	-0.11	0.19	0.11
5	-0.15	-0.22	0.12	0.07

Parameters of JR-5 to JR-8

Worker	β	γ	τ	θ
1	-0.20	-0.12	0.15	0.09
2	-0.23	-0.15	0.17	0.077
3	-0.19	-0.20	0.14	0.07
4	-0.30	-0.25	0.21	0.079
5	-0.21	-0.08	0.17	0.09
6	-0.22	-0.13	0.13	0.045
7	-0.18	-0.10	0.11	0.065
8	-0.25	-0.15	0.21	0.087
9	-0.18	-0.21	0.11	0.066
10	-0.17	-0.27	0.12	0.11

Parameters of JR-9 to JR-12

Worker	β	γ	τ	θ
1	-0.27	-0.18	0.14	0.07
2	-0.17	-0.23	0.15	0.08
3	-0.28	-0.21	0.13	0.08
4	-0.27	-0.13	0.20	0.10
5	-0.31	-0.26	0.17	0.08
6	-0.30	-0.22	0.12	0.12
7	-0.18	-0.09	0.19	0.09
8	-0.25	-0.19	0.18	0.09
9	-0.29	-0.16	0.12	0.08
10	-0.21	-0.09	0.19	0.07
11	-0.25	-0.18	0.20	0.10
12	-0.17	-0.23	0.15	0.12
13	-0.18	-0.15	0.15	0.10
14	-0.14	-0.21	0.18	0.09
15	-0.15	-0.10	0.11	0.11

Parameters of JR-13 to JR-16

Worker	β	γ	τ	θ
1	-0.20	-0.12	0.15	0.09
2	-0.23	-0.15	0.17	0.08
3	-0.19	-0.20	0.14	0.07
4	-0.30	-0.25	0.21	0.08
5	-0.21	-0.08	0.17	0.09
6	-0.22	-0.13	0.13	0.05
7	-0.18	-0.10	0.11	0.07
8	-0.25	-0.15	0.21	0.09
9	-0.18	-0.21	0.11	0.07
10	-0.17	-0.27	0.12	0.11
11	-0.33	-0.14	0.25	0.14
12	-0.24	-0.18	0.13	0.08
13	-0.16	-0.09	0.16	0.08
14	-0.27	-0.22	0.27	0.07
15	-0.26	-0.24	0.20	0.06
16	-0.29	-0.78	0.19	0.08
17	-0.20	-0.09	0.18	0.08
18	-0.17	-0.16	0.12	0.12
19	-0.18	-0.09	0.17	0.13
20	-0.19	-0.25	0.22	0.07

Appendix C

Derivation and Proof, Eq. (6.7)

Consider a string that represents a solution with n jobs and m machines. To generate a neighbouring solution, assume that the first operation of job one, 1^1, is selected. This operation could be swapped with any other jobs operation on the string with the exception of job one. Therefore, the number of possible neighbouring solutions that can be generated by swapping the position of job one operations could be mathematically presented as follow

$$1^1 : \left[(mn-1)-(m-1)\right] = (mn-m)$$
$$1^2 : \left[(mn-1)-(m-1)\right] = (mn-m)$$
$$1^3 : \left[(mn-1)-(m-1)\right] = (mn-m)$$
$$\vdots$$
$$1^m : \left[(mn-1)-(m-1)\right] = (mn-m)$$

The total number of possible neighbouring solutions that could be generated by swapping the position of any operation of job one is equal to the sum of the above expressions i.e.,

$$m(mn-m)$$

Now consider job two on the same string, each operation of job two can be swapped with operations of other jobs excluding the other operations of job two and job one operations (they have been already counted in the previous step). Therefore, the number of possible neighbouring solutions that could be generated by exchanging the position of each operation of job two with other operations on the string could be mathematically expressed as follows

Appendix C: Derivation and proof, Eq. (6.7)

$$2^1 : \left[(mn-1)-(m-1)-m\right] = (mn-2m)$$
$$2^2 : \left[(mn-1)-(m-1)-m\right] = (mn-2m)$$
$$2^3 : \left[(mn-1)-(m-1)-m\right] = (mn-2m)$$
$$\vdots$$
$$2^m : \left[(mn-1)-(m-1)-m\right] = (mn-2m)$$

Similar to that of job one, the total number of possible neighbouring solutions that could be generated by swapping the position of any operation of job two is the sum of the above expressions i.e.,

$$m(mn-2m)$$

Similarly for jobs k and n-1, the total number of neighbouring solutions that can be generated by swapping the position of these two jobs operations with other jobs operation on the string are respectively

$$m(mn-km)$$

and

$$m(mn-(n-1)m) = m^2$$

Therefore, the size of the neighbourhood of a solution with n jobs and m machines is calculated by adding up the number of neighbouring solutions that can be generated by swapping the position of each job operation on the string.

$$m(mn-m)+m(mn-2m)+m(mn-3m)+\ldots+m(mn-km)+\ldots+m^2$$

Simplifying the above equation yields

$$\text{Neighbourhood size} = m^2\left(\sum_{k=1}^{n-1}(n-k)\right)$$

Appendix D

Temporal Complexity Analysis

D.1 Complexity analysis of the SAMED-JSS

According to Aart and Van Laarhoven (1985), the computational complexity of a simulated annealing algorithm with cooling schedule presented in equation (6) is

$O(\tau L \ln|R|)$

where τ is the time to generate and possibly accept a solution, L is the length of the Markov chains, and R is the set of configurations or solutions.

For the case of job shop scheduling, the maximum number of steps required to generate and evaluate a solution has the complexity of $O(mn)$, i.e., $\tau \in O(mn)$ with m and n representing the number of machines and the number of jobs, respectively.

Aart and Van Laarhoven (1985) also showed that L, the length of the Markov chain, could be given by the maximum of the size of the solution sub-spaces. With the definition of the Markov chain, the complexity of L in the case of job shop scheduling could be presented as $L \in O(mn)$. Solution space for the job shop scheduling problem is $(n!)^m$, then R has the complexity of $R \in O((n!)^m)$. Therefore, the computational complexity of the simulated annealing algorithm for the job shop scheduling problem can be expressed as

$O((mn)^3(\ln n))$

Operations involving adding a neighbour to the tabu list and updating the seed memory list can be performed in $O(1)$. The GA component of the SAMED-JSS includes a random

selection mechanism and a crossover operator. Therefore, the GA component has the complexity of

$O(L_{ms}\, mn)$

where L_{ms} is the size of the long-term memory (or the population size).

In view of the above, the time complexity of the SAMED-JSS is calculated as follow:

Step 1 in SAMED-JSS algorithm has the complexity of $O(mn)$. Steps 2 and 3 have the complexity of $O((mn)^3 \ln n)$. Step 4 of the algorithm is executed only occasionally; therefore, it has the complexity of

$P^{GA}\, O(L_{ms}\, mn)$

where P^{GA} is the probability that GA component is called.

As a result, the complexity of the SAMED-JSS algorithm can be presented as:

$O((mn)^3 (\ln n)(L_{ms} S_{ms})) + P^{GA} O(L_{ms} mn)$ per long iteration

where S_{ms} is the size of the short-term memory.

In the above complexity expression, the second term is clearly of lower complexity than the first term; therefore, the overall complexity of the SAMED-JSS can be expressed as:

$O((mn)^3 (\ln n)(L_{ms} S_{ms}))$

D.2 Complexity analysis of the SAMED-LB

To calculate the time complexity of the SAMED-LB, it is required to determine the complexity of the local search. The complexity of the local search component of the SAMED-LB is calculated as follows.

Finding and storing the critical path (critical blocks) of a solution is performed at the complexity of $O(mn)$.

Assuming that the number of operations on a critical path is S, blocks of operations and set of moves then can be performed in $O(S)$. Makespan of each neighbour generated by swap operation is calculated in $O(mn)$. As a result, the computational complexity of the local search component is

Temporal complexity analysis

$O(qmn)$

where q is the number of neighbours on the critical path.

The operations involving blockage removal can be performed in $O(1)$. Having the computational complexity of the SAMED and those of the above two components, i.e., local search and blockage removal, the temporal complexity of the SAMED-LB is calculated as

$$O(((mn)^3(\ln n)+(qmn/S_{ms}))L_{ms}S_{ms})+P^{GA}O(L_{ms}mn)$$

The above equation could be further simplified and presented as

$$O((mn)^3(\ln n)(L_{ms}S_{ms})+(qmn)L_{ms})+P^{GA}O(L_{ms}mn) \text{ per long iteration}$$

Similar to that of the SAMED algorithm, the complexity of the second term in the above expression is clearly lower than that of the first term; therefore, the overall complexity of the SAMED-LB is in the order of:

$$O((mn)^3(\ln n)(L_{ms}S_{ms})+(qmn)L_{ms})$$

Appendix E

Sample Computer Program Codes

E.1 Job shop scheduling by SAMED-LB

Main module:

```
Public Type Idx
Row As Integer
Col As Integer
End Type
Public TempO As Idx
Public Tbct() As Integer
Public TheBstsolution() As Integer
Public ReplictionCount As Integer
Public Makspans() As Integer
Public Solutnfound() As Integer
Public replicationNo As Integer
Public Cromosm() As Integer
Public CromosmIndx() As Integer
Public PopCromIndx() As Integer
Public IndivCromosmIndx() As Integer
Public PrevusItrnBstCoTime As Integer
Public Fcount As Integer
Public BstMvCotime As Integer
Public Startime As Double
Public memsize As Integer
Public Nowakineded As Boolean
Public prvCoTmin As Integer
Public Movenumbr As Integer
Public ItrCount As Long
Public Prnt1() As Integer
Public Prnt1Indx() As Integer
Public Prnt2() As Integer
Public Prnt2Indx() As Integer
Public BstMvCromosm() As Integer
Public OfspCromosm() As Integer
Public OfspnIndx() As Integer
Public RandGen2 As Integer
Public RandGen1 As Integer
Public PopCromosm() As Integer
Public PopCotime() As Integer
Public OfsCotime() As Integer
Public Popmem As Integer
Public Pnumbr As Integer
Public SelctdGen1 As Integer
Public SelctdGen2 As Integer
Public TmpBstSchedule() As Integer
Public taboList() As Integer, tabuHits As Long
Public mtaboList() As Integer
```

```
Public NofTabu As Integer, tabuIdx As Integer, tabuFilled As Boolean
Public scunt As Long
Public r As Integer
Public njob As Integer
Public nMachine As Integer
Public JD() As Integer
Public JJ() As Integer
Public DT() As Integer
Public MD() As Integer
Public CT() As Integer
Public BstSchedule() As Integer
Public CTemptur As Currency
Public ItrnBschl() As Integer
Public CoTime As Integer
Public CoTmin As Integer
Public ItrnBstCoTime As Integer
Public bMinTime As Long
Public MaxNofrmoves As Long
Public Temtur As Currency
Public m As Long
Public nebvalid As Boolean
Public mvinItern As Integer
Public Cnbhd As Integer
Public Moveno1 As Integer
'read data pertaining to job duration times
Sub read_data2(DFname2 As String)
On Error GoTo ShErr:
Open DFname2 For Input As #1
Dim i As Integer
Input #1, njob
Input #1, nMachine
ReDim DT(1 To njob, 1 To nMachine) As Integer
For i = 1 To njob
        For k = 1 To nMachine
    Input #1, DT(i, MD(i, k)) ' Read data to array
      Next k
  Next i
Close #1
Exit Sub
ShErr:
  MsgBox "Error in reading from file \n Check file name and path."
End Sub
'Read data pertaining to operations constraint of each job
Sub read_data1(DFname1 As String)
On Error GoTo ShErr:
Open DFname1 For Input As #1
Dim j As Integer
Input #1, njob
Input #1, nMachine
nMachine = nMachine - 1
ReDim MD(1 To njob, 0 To nMachine) As Integer
For j = 1 To njob
  For k = 0 To nMachine
    Input #1, MD(j, k) ' Read data to array
  Next k
Next j
Close #1
Exit Sub
ShErr:
  MsgBox "Error in reading from file \n Check file name and path."
End Sub
```

```
Sub read_data3(DFname3 As String)
On Error GoTo ShErr:
Open DFname3 For Input As #1
Input #1, njob
Input #1, nMachine
ReDim Cromosm(1 To njob * nMachine) As Integer

        For j = 1 To njob * nMachine
          Input #1, Cromosm(j)  ' Read data to array

        Next j
Close #1
Exit Sub
ShErr:
  MsgBox "Error in reading from file \n Check file name and path."
End Sub

Sub read_data4(DFname4 As String)
On Error GoTo ShErr:
Open DFname4 For Input As #1
Dim i As Integer
Input #1, njob
Input #1, nMachine
ReDim CromosmIndx(1 To njob * nMachine) As Integer

        For j = 1 To njob * nMachine

         Input #1, CromosmIndx(j)  ' Read data to array

        Next j
Close #1
Exit Sub
ShErr:
  MsgBox "Error in reading from file \n Check file name and path."
End Sub

Sub WritData1(OutptFile1 As String)
Open OutptFile1 For Output As #1
Dim i As Integer
Dim changeLine As Boolean
changeLine = False
   For i = 1 To njob * nMachine

       Print #1, Cromosm(i);          ' Read data to array
   Next i

Close #1
Exit Sub
ShErr:
  MsgBox "Error in reading from file \n Check file name and path."
End Sub

Sub WritData2(OutptFile2 As String)
Open OutptFile2 For Output As #1
Dim i As Integer
Dim changeLine As Boolean
changeLine = False
   For j = 1 To njob * nMachine

       Print #1, CromosmIndx(j);          ' Read data to array

   Next j
```

```
Close #1
Exit Sub
ShErr:
  MsgBox "Error in reading from file \n Check file name and path."
End Sub

Sub WritData3(OutptFile3 As String)
Open OutptFile3 For Output As #1
Dim i As Integer
Dim changeLine As Boolean
changeLine = False
For i = 1 To njob
   For j = 1 To nMachine

       If j = nMachine Then
              changeLine = True
        Print #1, Tbct(i, j)
          changeLine = False
       Else
         Print #1, Tbct(i, j);          ' Read data to array
       End If
   Next j
   changeLine = False
Next i
Close #1
Exit Sub
ShErr:
  MsgBox "Error in reading from file \n Check file name and path."
End Sub

Sub WritData4(OutptFile4 As String)
On Error GoTo ShErr:
Open OutptFile4 For Output As #1
Dim i As Integer
'
For i = 0 To ReplictionCount

        Print #1, Makspans(i), Solutnfound(i)
Next i
Close #1
Exit Sub
ShErr:
  MsgBox "Error in writing to file OutptFile4\ Check file name and path."
End Sub

Sub RandPopGenerator()
Randomize Time
Dim Gene As Integer
Gene = 0
Dim GeneLoctn As Integer
GeneLoctn = 0
Dim RandnoValid As Boolean
RandnoValid = True
Dim j As Integer
j = 0
ReDim jobId(1 To njob) As Integer
ReDim JobIndex(1 To njob) As Integer
ReDim GeneLocation(1 To njob * nMachine)
ReDim Cromosm(1 To njob * nMachine)
ReDim RandCromosm(1 To njob * nMachine)
For i = 1 To njob
jobId(i) = i
```

```
Next i
ReDim OprtionReption(1 To njob) As Integer
ReDim GeneOptIndx(1 To njob * nMachine)
ReDim CromosmIndx(1 To njob * nMachine)
ReDim RandCromosmIndx(1 To njob * nMachine)

        Do

            RandNum = Int((njob) * Rnd + 1)   'Generate a random number between 1 and number of jobs

                Do
                  j = j + 1
                    If jobId(j) = RandNum Then  ' check if RndN is job i
                      If JobIndex(j) = nMachine Then 'check if the number of job i generated is equal to the number
of operations

                          RandnoValid = False

                        Else
                          JobIndex(j) = JobIndex(j) + 1  'Calculate the number of job(s) i generated so far
                          Gene = Gene + 1
                          GeneLocation(Gene) = jobId(j) 'put the generated job on the k'th location of cromosom
                          j = njob
                        End If

                    End If

              Loop Until j = njob Or RandnoValid = False
                j = 0
                 RandnoValid = True
          Loop While Gene < njob * nMachine
                Gene = 0
                For i = 1 To (njob * nMachine)
                  RandCromosm(i) = GeneLocation(i)
                Next i

      Do

          k = k - 1

              Do
                  j = j + 1
                  If jobId(j) = RandCromosm(k) Then 'check if the k'th member of cromosom is job j

                    OprtionReption(j) = OprtionReption(j) + 1 'calculate the number of repetion of job j at the
coromosom
                    GeneLoctn = GeneLoctn + 1

                    GeneOptIndx(GeneLoctn) = OprtionReption(j) 'assign the number of repetion (operation
no.)of job j at cromosom
                    j = njob

                  End If

            Loop Until j = njob

             j = 0

      Loop While GeneLoctn < (njob * nMachine)

For i = 1 To (njob * nMachine)
```

```
        RandCromosmIndx(i) = GeneOptIndx(i)
Next i
For i = 1 To (njob * nMachine)
        CromosmIndx(i) = RandCromosmIndx(i)
       Cromosm(i) = RandCromosm(i)

Next i
ReDim JobIndex(1 To njob) As Integer
ReDim GeneLocation(1 To njob * nMachine)
ReDim OprtionReption(1 To njob) As Integer
ReDim GeneOptIndx(1 To njob * nMachine)
k = 0
GeneLoctn = 0
End Sub

Public Sub Schedul()
Dim tempComTime As Integer, EndTime As Integer, L As Integer
Dim succss As Boolean
Dim schedl As Boolean
Dim a As Integer
Dim b As Integer
Dim e As Integer
ReDim machinTime(1 To nMachine)
ReDim jobTime(1 To njob)
ReDim JD(0 To njob * nMachine)
'schduling the jobs on machines and computing the complition time
For i = 1 To njob * nMachine
   JD(i) = Cromosm(i)
Next i
ReDim CT(0 To njob, 0 To nMachine)
For i = 1 To njob
   MD(i, 0) = 0
Next i
For i = 1 To njob * nMachine

        CT(JD(i), 0) = 1
Next i
JD(0) = 0
For L = 1 To nMachine
   CT(0, L) = 1
Next L
nebvalid = True
e = 0
t = 0
tempComTime = 0
EndTime = 0
Do
   t = t + 1
   i = 0
   k = 0

      Do
         i = i + 1
         a = JD(i)
         Do
            k = k + 1
            b = MD(a, k)

               If CT(JD(i), MD(a, k)) = 0 Then
                  If CT(JD(i), MD(a, k - 1)) > 0 Then
                        If jobTime(JD(i)) > machinTime(MD(a, k)) Then
                           CT(JD(i), MD(a, k)) = jobTime(JD(i)) + DT(JD(i), MD(a, k))
```

```
                    succss = True
                    e = e + 1
                    tempComTime = CT(JD(i), MD(a, k))
                    jobTime(JD(i)) = CT(JD(i), MD(a, k))
                    machinTime(MD(a, k)) = CT(JD(i), MD(a, k))
                    If EndTime < tempComTime Then EndTime = tempComTime
                  Else
                    CT(JD(i), MD(a, k)) = machinTime(MD(a, k)) + DT(JD(i), MD(a, k))
                    succss = True
                    e = e + 1
                    tempComTime = CT(JD(i), MD(a, k))
                    jobTime(JD(i)) = CT(JD(i), MD(a, k))
                    machinTime(MD(a, k)) = CT(JD(i), MD(a, k))
                    If EndTime < tempComTime Then EndTime = tempComTime

                End If
              Else
               succss = True
               End If
             End If

        Loop Until k = nMachine Or succss = True Or schedl = True
        schedl = False
         succss = False
        k = 0
      Loop Until i = njob * nMachine
      succss = False
      i = 0
  Loop Until e = njob * nMachine
CoTime = EndTime
End Sub

Sub nextMove()
Randomize Time
Dim CoTdf As Integer
Dim EXx As Double
Dim F As Currency
Dim MRdn As Currency
Dim Success As Boolean
Dim PopNumbr As Integer
If tabuFilled = True Then
   CoTime = ItrnBstCoTime
End If
Dim ccCoTime As Integer
Dim nofmov As Integer
Cnbhd = CoTime
tabuFilled = False
isTrap = False
Do

   selctneibour
   Schedul
   CoTdf = CoTime - Cnbhd
   F = CoTdf / CTemptur
   EXx = Exp(-F)
   If EXx >= 1 Then
    Success = True
   Else
     MRdn = Rnd
     If MRdn < EXx Then
     Success = True
     Else
```

```
        Resetneibour
        ccCoTime = CoTime
        CoTime = Cnbhd
        nofmov = nofmov + 1
         If nofmov > MaxNofrmoves Then
             Success = True
              Resetneibour
            CoTime = ccCoTime
          End If
      End If
   End If
Loop Until Success
nofmov = 0
addJobtoTabu

If tabuFilled = True Then
        Movenumbr = 1
          If ItrCount <> 0 Then
              If CoTime < BstMvCotime Then
                 BstMvCotime = CoTime
                 For i = 1 To njob * nMachine
                  BstMvCromosm(i) = Cromosm(i)
                 Next i
               End If
            Schdneded = False
          End If

          If CoTime <= ItrnBstCoTime Then
             ItrnBstCoTime = CoTime
              ItrnBestSchdul
                Bstsultion
          Else
             Schdneded = True
             Bstsultion
          End If

          If ItrCount = 0 Then
                Popmem = Popmem + 1

                 If Popmem = memsize + 1 Then
                    ItrCount = ItrCount + 1
                    prvCoTmin = CoTmin
                    PrevusItrnBstCoTime = ItrnBstCoTime
                    Popmem = 1
                 End If

          Else

              If prvCoTmin = CoTmin Then

                        If PrevusItrnBstCoTime = ItrnBstCoTime Then

                                 For i = 1 To njob * nMachine

                                     PopCromosm(Popmem, i) = BstMvCromosm(i)
                                 Next i

                                     PopCromosmIndx
                                     Popmem = Popmem + 1

                                   If Popmem = memsize + 1 Then
```

```
                    ItrCount = ItrCount + 1
                    GAComponent
                    prvCoTmin = CoTmin
                    PrevusItrnBstCoTime = ItrnBstCoTime
                    Popmem = 1

                End If

            Else

                If Schdneded Then
                  Schedul
                End If
                   selctneibourNwki

                   If ItrnBstCoTime < CoTmin Then
                           CoTmin = ItrnBstCoTime
                           For i = 1 To njob
                                For j = 1 To nMachine
                                    BstSchedule(i, j) = TmpBstSchedule(i, j)
                                Next j
                            Next i
                           bMinTime = Timer - Startime
                           Moveno1 = bMinTime
                          Bstsultion
                        CoTime = ItrnBstCoTime
                       Popmem = Popmem + 1
                      If Popmem = memsize + 1 Then
                            ItrCount = ItrCount + 1
                            prvCoTmin = CoTmin
                            Popmem = 1
                            PrevusItrnBstCoTime = ItrnBstCoTime

                        End If

                   Else
                          Bstsultion

                             For i = 1 To njob * nMachine

                               PopCromosm(Popmem, i) = Cromosm(i)

                             Next i

                            PopCromosmIndx
                            Popmem = Popmem + 1
                            If Popmem = memsize + 1 Then
                                   ItrCount = ItrCount + 1
                                     GAComponent
                                    prvCoTmin = CoTmin
                                   PrevusItrnBstCoTime = ItrnBstCoTime
                                      Popmem = 1
                            End If

                End If


            End If

        Else

            If Schdneded Then
```

```
                    Schedul
                End If

                selctneibourNwki

                If ItrnBstCoTime < CoTmin Then
                    CoTmin = ItrnBstCoTime
                        For i = 1 To njob
                            For j = 1 To nMachine
                                BstSchedule(i, j) = TmpBstSchedule(i, j)
                            Next j
                        Next i

                    bMinTime = Timer - Startime

                        Moveno1 = bMinTime
                        Bstsultion

                    CoTime = ItrnBstCoTime
                Else
                    Bstsultion
                End If
            Popmem = Popmem + 1
                If Popmem = memsize + 1 Then
                    ItrCount = ItrCount + 1
                    prvCoTmin = CoTmin
                    Popmem = 1
                    PrevusItrnBstCoTime = ItrnBstCoTime
                End If

        End If

    End If

Else

    If ItrCount <> 0 Then
        If Movenumbr = 1 Then
            BstMvCotime = CoTime
                For i = 1 To njob * nMachine

                    BstMvCromosm(i) = Cromosm(i)

                Next i
        Else

            If CoTime < BstMvCotime Then
                BstMvCotime = CoTime

                    For i = 1 To njob * nMachine

                        BstMvCromosm(i) = Cromosm(i)

                    Next i
            End If
        End If
        Movenumbr = Movenumbr + 1

    End If

        If CoTime <= ItrnBstCoTime Then
            ItrnBstCoTime = CoTime
```

```
                ItrnBestSchdul

            End If

End If
End Sub

Sub SimulationSearch(maxMove As Integer)
    ReDim Tbct(1 To njob, 1 To nMachine)
    ReDim TheBstsolution(1 To njob)
    ReDim Makspans(0 To 30)
    ReDim Solutnfound(0 To 30)
    ReplictionCount = 0
Do While ReplictionCount < replicationNo
        RandPopGenerator
      NofTabu = ViewDFrm.tabuLength.Text
      memsize = ViewDFrm.msize.Text
    ReDim taboList(1 To njob * nMachine) As Integer
    tabuIdx = 1
      tabuFilled = False
     Schedul
     r = 0
    Dim SearchTime As Double, Start As Double, Finish As Double, MinTime As Double
     Dim TotlCoTime As Long
     Dim MinCoTime As Currency
     Dim DevCoTime As Currency
     Dim TotDevCoTime As Currency
     Dim SDCoTime As Currency
     Dim Delta As Currency
     Start = Timer
     Startime = Timer
     SearchTime = ViewDFrm.MaxMov.Text
     MinTime = Start
     Dim icount As Long
     mvinItern = 0
     Pnumbr = 1
     CoTmin = CoTime
     ItrnBstCoTime = CoTime
     ItrCount = 0
     TotlCoTime = CoTime
     ReDim nComTime(0 To 100000) As Long
     nComTime(0) = CoTime
     CTemptur = 0.95
     Delta = 0.1
     icount = 0
     m = 1
     S = 1
    Popmem = 1
    Movenumbr = 1
    ReDim BstMvCromosm(1 To njob * nMachine)
    ReDim OfspCromosm(1 To memsize, 1 To njob * nMachine)
    ReDim PopCromosm(1 To memsize, 1 To njob * nMachine)
    ReDim PopCromIndx(1 To memsize, 1 To njob * nMachine)
    ReDim PopCotime(1 To memsize)
    ReDim OfsCotime(1 To memsize)
    Do While Timer < Start + SearchTime
    MaxNofrmoves = (nMachine * ((njob * (njob - 1) / 2))) / 6
     nextMove
     If m = 1 Then
       CTemptur = 0.95
    Else
        icount = icount + 1
```

```
        nComTime(icount) = CoTime
        DevCoTime = 0
        TotDevCoTime = 0
        SDCoTime = 0
        TotlCoTime = TotlCoTime + CoTime
        MinCoTime = TotlCoTime / (m + 1)

      For icount = 0 To m
        DevCoTime = ((nComTime(icount) - MinCoTime) ^ 2) / (m + 1)
        TotDevCoTime = TotDevCoTime + DevCoTime
      Next icount
        SDCoTime = Sqr(TotDevCoTime)
        CTemptur = CTemptur / (1 + (CTemptur * Log(1 + Delta) / (3 * SDCoTime)))
      If CTemptur < 0.000001 Then CTemptur = 0.000001

    End If
      If CoTime < CoTmin Then
          Nowakineded = False
          CoTmin = CoTime
          BestSchdul
          bMinTime = Timer - Start
          Moveno1 = bMinTime
      End If
       If (m Mod 10) = 0 Then
            ViewDFrm.caLabel.Caption = "Calculating...." & CoTmin
            ViewDFrm.caLabel.Refresh
       End If
      m = m + 1
    Loop
 If CoTmin <= 784 Then

          Dim h As Integer
          Dim j As Integer

          For h = 1 To njob
              For j = 1 To nMachine
                 Tbct(h, j) = BstSchedule(h, j)

              Next j

          Next h

    End If

    Makspans(ReplictionCount) = CoTmin
    Solutnfound(ReplictionCount) = Moveno1
    ReplictionCount = ReplictionCount + 1
 Loop
End Sub

Public Sub selctneibour()
Randomize Time
Dim TempGen As Integer
Succs = False
Dim Temp As Integer
  Dim inTabu As Boolean
  Dim nofTry As Integer, maxTry As Integer
  inTabu = True
  nofTry = 0
  maxTry = 100
Do While inTabu
    RandGen1 = Int((njob * nMachine) * Rnd + 1)
```

```
        Do While Succs = False
          RandGen2 = Int((njob * nMachine) * Rnd + 1)
          If RandGen2 <> RandGen1 Then
            If Cromosm(RandGen1) <> Cromosm(RandGen2) Then
                Succs = True
            End If
          End If
        Loop
      inTabu = isJobinTabu
    If inTabu Then ' If it is in tabu list then return to previous state
      nofTry = nofTry + 1
      Succs = False
      If nofTry > maxTry Then
        inTabu = False

      End If
    End If

Loop

TempGen = Cromosm(RandGen2)
Cromosm(RandGen2) = Cromosm(RandGen1)
Cromosm(RandGen1) = TempGen
End Sub

Public Sub selctneibourNwki()
Randomize Time
IndividulCrmosmIndx
ReDim TmpBstSchedule(1 To njob, 1 To nMachine)
ReDim TempCT(1 To njob, 1 To nMachine)
ReDim TempCromosm(1 To njob * nMachine)
ReDim CrBlock(1 To (10 * nMachine), 1 To njob)
ReDim CrmachNo(1 To (10 * nMachine))
ReDim LstJbinCrBlk(1 To (10 * nMachine))
ReDim OprtionNo(1 To (10 * nMachine), 1 To 600)
ReDim NofCrjbonBlk(1 To (10 * nMachine))
ReDim NwakiBstCromosm(1 To njob * nMachine)
Dim CrtclPathFond As Boolean
Dim BreakPintFound As Boolean
Dim Nsucess As Boolean
Dim NNsucess As Boolean
Dim NNNsucess As Boolean
Dim OprtnNoFound As Boolean
Dim Endblck As Boolean
ReDim TempIndivCromosmIndx(1 To njob * nMachine)
Dim ClmnMxCT As Integer
Dim machNo As Integer
Dim q As Integer
Dim MxCT As Integer
Dim jbNo As Integer
Dim RdCrGen1 As Integer
Dim RdCrGen2 As Currency
Dim MachCunt As Integer
MxCT = CT(1, 1)
MachCunt = 1
CrtclPathFond = False
For i = 1 To njob
   For j = 1 To nMachine
     If CT(i, j) >= MxCT Then
       MxCT = CT(i, j)
       machNo = j
        jbNo = i
```

```
    End If

  Next j
Next i
q = 1
 CrBlock(q, 1) = jbNo
  CrmachNo(1) = machNo
 NofCrjbonBlk(q) = 0
Do While CrtclPathFond = False
          For i = 1 To njob
          TempCT(i, machNo) = CT(i, machNo)
           Next i

            If q >= 2 Then
              For i = 1 To njob
                If TempCT(i, machNo) > (CT(LstJbinCrBlk(q - 1), CrmachNo(MachCunt - 1)) -
DT(LstJbinCrBlk(q - 1), CrmachNo(MachCunt - 1))) Then
                   TempCT(i, machNo) = 0
                End If
              Next i

             End If
            For j = 1 To njob

                ClmnMxCT = TempCT(1, machNo)
                CrBlock(q, j) = 1
              For i = 1 To njob
                If TempCT(i, machNo) > ClmnMxCT Then
                  ClmnMxCT = TempCT(i, machNo)
                  CrBlock(q, j) = i
                End If

              Next i
               TempCT(CrBlock(q, j), machNo) = 0
            Next j

    OprtnNoFound = False
    i = 1
    Do While i < njob + 1
      L = 1
          Do While OprtnNoFound = False
              If MD(CrBlock(q, i), L) = machNo Then
                   OprtionNo(q, CrBlock(q, i)) = L
                    OprtnNoFound = True
              End If
            L = L + 1
          Loop
          i = i + 1
          OprtnNoFound = False
   Loop
   i = 2
    BreakPintFound = False
  Do While BreakPintFound = False  'determine the critical jobs on the block(i.e.,determine the borders of the critical
block on a machine

        NofCrjbonBlk(q) = NofCrjbonBlk(q) + 1
        'If CT(CrBlock(machNo, i), machNo) <> CT(CrBlock(machNo, i - 1), machNo) - DT(CrBlock(machNo, i - 1),
OprtionNo(CrBlock(machNo, i - 1))) Then
        If CT(CrBlock(q, i), machNo) <> CT(CrBlock(q, i - 1), machNo) - DT(CrBlock(q, i - 1), machNo) Then
            BreakPintFound = True
            For k = i To njob
            CrBlock(q, k) = 0
```

```
            Next k
            machNo = MD(CrBlock(q, i - 1), (OprtionNo(q, CrBlock(q, i - 1))) - 1) 'determine the Predecessor
operation of the last(first) job on the block
            'machNo = MD(CrBlock(machNo, i), OprtionNo(CrBlock(machNo, i)) - 1) 'determine the Predecessor
operation of the last(first) job on the block
            MachCunt = MachCunt + 1
            CrmachNo(MachCunt) = machNo
          LstJbinCrBlk(q) = CrBlock(q, i - 1)
        End If
      i = i + 1
      If i = njob + 1 And BreakPintFound = False Then
         machNo = MD(CrBlock(q, i - 1), (OprtionNo(q, CrBlock(q, i - 1))) - 1)
         MachCunt = MachCunt + 1
         CrmachNo(MachCunt) = machNo
          BreakPintFound = True
         LstJbinCrBlk(q) = CrBlock(q, i - 1)

      End If
    Loop
      q = q + 1
  If machNo = 0 Then
   CrtclPathFond = True
  q = q - 1
 End If
Loop

   Dim Fcount As Integer
   Fcount = 0
   Rcount = 1
 Do While Rcount < q + 1

   If NofCrjbonBlk(Rcount) > 1 Then
          If Rcount = 1 Then

                    Nsucess = False
                      k = 1
                      Do While k < (njob * nMachine) + 1 And Nsucess = False
                          If CrBlock(Rcount, NofCrjbonBlk(Rcount)) = Cromosm(k) Then
                             If OprtionNo(Rcount, CrBlock(Rcount, NofCrjbonBlk(Rcount))) = IndivCromosmIndx(k)
Then

                                Nsucess = True
                                SelctdGen1 = k
                             End If
                          End If
                        k = k + 1
                      Loop

                      Nsucess = False
                      u = 1
                      Do While u < (njob * nMachine) + 1 And Nsucess = False

                          If CrBlock(Rcount, NofCrjbonBlk(Rcount) - 1) = Cromosm(u) Then
                             If OprtionNo(Rcount, CrBlock(Rcount, NofCrjbonBlk(Rcount) - 1)) =
IndivCromosmIndx(u) Then
                                Nsucess = True
                                SelctdGen2 = u
                             End If
                          End If
                        u = u + 1
                      Loop
```

```
        Else

            If Rcount = q Then

                Nsucess = False
                k = 1
                Do While k < (njob * nMachine) + 1 And Nsucess = False

                    If CrBlock(q, 1) = Cromosm(k) Then
                      If OprtionNo(q, CrBlock(q, 1)) = IndivCromosmIndx(k) Then
                          Nsucess = True
                          SelctdGen1 = k
                       End If
                    End If

                    k = k + 1
                Loop

                  Nsucess = False
                    u = 1
                 Do While u < (njob * nMachine) + 1 And Nsucess = False

                      If CrBlock(q, 2) = Cromosm(u) Then
                          If OprtionNo(q, CrBlock(q, 2)) = IndivCromosmIndx(u) Then

                             Nsucess = True
                             SelctdGen2 = u
                          End If
                      End If

                      u = u + 1
                Loop

            Else

                RdCrGen2 = Rnd
                If RdCrGen2 <= 0.5 Then
                    Nsucess = False
                      k = 1
                    Do While k < (njob * nMachine) + 1 And Nsucess = False

                        If CrBlock(Rcount, NofCrjbonBlk(Rcount)) = Cromosm(k) Then
                          If OprtionNo(Rcount, CrBlock(Rcount, NofCrjbonBlk(Rcount))) =
IndivCromosmIndx(k) Then

                              Nsucess = True
                              SelctdGen1 = k
                           End If
                        End If

                        k = k + 1
                      Loop

                        Nsucess = False
                          u = 1
                       Do While u < (njob * nMachine) + 1 And Nsucess = False

                          If CrBlock(Rcount, NofCrjbonBlk(Rcount) - 1) = Cromosm(u) Then
                            If OprtionNo(Rcount, CrBlock(Rcount, NofCrjbonBlk(Rcount) - 1)) =
IndivCromosmIndx(u) Then
                                Nsucess = True
                                SelctdGen2 = u
```

```
                    End If
                End If
            u = u + 1
        Loop

    Else

        Nsucess = False
            k = 1
            Do While k < (njob * nMachine) + 1 And Nsucess = False

                If CrBlock(Rcount, 1) = Cromosm(k) Then
                    If OprtionNo(Rcount, CrBlock(Rcount, 1)) = IndivCromosmIndx(k) Then

                        Nsucess = True
                        SelctdGen1 = k
                    End If
                End If

                k = k + 1
            Loop

            Nsucess = False
                u = 1
            Do While u < (njob * nMachine) + 1 And Nsucess = False

                If CrBlock(Rcount, 2) = Cromosm(u) Then
                    If OprtionNo(Rcount, CrBlock(Rcount, 2)) = IndivCromosmIndx(u) Then
                        Nsucess = True
                        SelctdGen2 = u
                    End If
                End If

                u = u + 1
            Loop

    End If

        End If
    End If

TemppCm = Cromosm(SelctdGen1)
TemppCmIndx = IndivCromosmIndx(SelctdGen1)
Cromosm(SelctdGen1) = Cromosm(SelctdGen2)
IndivCromosmIndx(SelctdGen1) = IndivCromosmIndx(SelctdGen2)
Cromosm(SelctdGen2) = TemppCm
IndivCromosmIndx(SelctdGen2) = TemppCmIndx
 Schedul

    Fcount = Fcount + 1
    If Fcount = 1 Then
     NowakiBestCoTime = CoTime
       For i = 1 To njob * nMachine
          NwakiBstCromosm(i) = Cromosm(i)
       Next i
       For i = 1 To njob
            For j = 1 To nMachine
                TmpBstSchedule(i, j) = CT(i, j)
            Next j
      Next i

    Else
```

```
            If CoTime < NowakiBestCoTime Then
                NowakiBestCoTime = CoTime
              For i = 1 To njob * nMachine
                NwakiBstCromosm(i) = Cromosm(i)
              Next i

              For i = 1 To njob
                 For j = 1 To nMachine
                   TmpBstSchedule(i, j) = CT(i, j)
                 Next j
              Next i

            End If

        End If

        ResetneibourNwki

    End If

    Rcount = Rcount + 1

Loop

        If NowakiBestCoTime < ItrnBstCoTime Then
            ItrnBstCoTime = NowakiBestCoTime

            For i = 1 To njob * nMachine
                Cromosm(i) = NwakiBstCromosm(i)
            Next i

            ItrnBestSchdul

        End If

End Sub

Public Sub ResetneibourNwki()
TempGen = Cromosm(SelctdGen1)
Cromosm(SelctdGen1) = Cromosm(SelctdGen2)
Cromosm(SelctdGen2) = TempGen
End Sub

Function isJobinTabu() As Boolean
Dim FoundInTabu As Boolean ', Equal As Boolean
FoundInTabu = False
If taboList(RandGen1) = Cromosm(RandGen1) Or taboList(RandGen2) = Cromosm(RandGen2) Then
FoundInTabu = True
End If
isJobinTabu = FoundInTabu
End Function

Sub addJobtoTabu()
taboList(RandGen1) = Cromosm(RandGen1)
taboList(RandGen2) = Cromosm(RandGen2)

  If tabuIdx = NofTabu Then
     ReDim taboList(1 To njob * nMachine) As Integer
     tabuIdx = 1
     tabuFilled = True ' changes first time that tabu has been filled
  Else
    tabuIdx = tabuIdx + 1
```

```
  End If
End Sub

Public Sub ItrnBestSchdul()
ReDim ItrnBschl(1 To njob * nMachine)
For i = 1 To njob * nMachine
   ItrnBschl(i) = Cromosm(i)
Next i
End Sub
Public Sub Bstsultion()
For i = 1 To njob * nMachine
    Cromosm(i) = ItrnBschl(i)
Next i
End Sub

Public Sub Resetneibour()
TempGen = Cromosm(RandGen2)
 Cromosm(RandGen2) = Cromosm(RandGen1)
 Cromosm(RandGen1) = TempGen
End Sub

Public Sub BestSchdul()
ReDim BstSchedule(1 To njob, 1 To nMachine)
For i = 1 To njob
   For j = 1 To nMachine
   BstSchedule(i, j) = CT(i, j)
   Next j
Next i
End Sub

Public Sub GAComponent()
Dim Bingoo As Boolean
 Randomize Time
  Dim RanndomN As Currency
   Pnumbr = 1
   Dim Noftrry As Integer

   Do While Pnumbr < memsize + 1
        ParntSelectnGDdd
         PPXCrosOver
        For i = 1 To njob * nMachine
            Cromosm(i) = OfspCromosm(Pnumbr, i)
        Next i
          Schedul
          OfsCotime(Pnumbr) = CoTime
        If CoTime <= ItrnBstCoTime Then
             ItrnBstCoTime = CoTime
              ItrnBestSchdul
        End If
    Pnumbr = Pnumbr + 1
   Loop

If ItrnBstCoTime < CoTmin Then
     Bstsultion
     CoTime = ItrnBstCoTime
Else

                    RanndomN = Rnd
                   If RanndomN < (1 - CTemptur) Then
                        CTemptur = 0.95
                        Noftrry = 1
                        Bingoo = False
```

```
                Do While Noftrry < (memsize + 1) And Bingoo = False
                    If ItrnBstCoTime <> OfsCotime(Noftrry) Then
                       Bingoo = True
                       For i = 1 To njob * nMachine
                         Cromosm(i) = OfspCromosm(Noftrry, i)
                       Next i
                         ItrnBstCoTime = OfsCotime(Noftrry)
                         CoTime = OfsCotime(Noftrry)
                         ItrCount = 0
                      End If
                      Noftrry = Noftrry + 1
                Loop
             Else
              Bstsultion
              CoTime = ItrnBstCoTime
             End If

End If
      If (scunt Mod 1) = 0 Then
      ViewDFrm.Ca2labl.Caption = "Calculating...." & ItrnBstCoTime
      ViewDFrm.Ca2labl.Refresh
      End If
     scunt = scunt + 1

End Sub
Sub ParntSelectnGDdd()
 Randomize Time
 Dim RandCrmsm1 As Integer
 Dim RandCrmsm2 As Integer
 Dim Sucsss As Boolean
 ReDim Prnt1(1 To njob * nMachine)
 ReDim Prnt2(1 To njob * nMachine)
 ReDim Prnt1Indx(1 To njob * nMachine)
 ReDim Prnt2Indx(1 To njob * nMachine)
 RandCrmsm1 = Int((memsize) * Rnd + 1)   'Select Parent 1 and Parent 2 randomly from the Population
 Sucsss = False
       Do While Sucsss = False
         RandCrmsm2 = Int((memsize) * Rnd + 1)
         If RandCrmsm1 <> RandCrmsm2 Then
            Sucsss = True
          End If
       Loop

   For j = 1 To njob * nMachine
    Prnt1(j) = PopCromosm(RandCrmsm1, j)
       Prnt1Indx(j) = PopCromIndx(RandCrmsm1, j)
       Prnt2(j) = PopCromosm(RandCrmsm2, j)
       Prnt2Indx(j) = PopCromIndx(RandCrmsm2, j)
  Next j
End Sub

Sub ParntSelectnGD()
 Randomize Time
 Dim RandCrmsm1 As Integer
 Dim RandCrmsm2 As Integer
 Dim Sucsss As Boolean
 ReDim Prnt1(1 To njob * nMachine)
 ReDim Prnt2(1 To njob * nMachine)
 ReDim Prnt1Indx(1 To njob * nMachine)
 ReDim Prnt2Indx(1 To njob * nMachine)
 RandCrmsm1 = Int((memsize) * Rnd + 1)   'Select Parent 1 and Parent 2 randomly from the Population
 Sucsss = False
```

```
 Prntunidentical = False
 noftryy = 0
         Do While Sucsss = False Or Prntunidentical = False
           noftryy = noftryy + 1
           RandCrmsm2 = Int((memsize) * Rnd + 1)
           If RandCrmsm1 <> RandCrmsm2 Then
                Sucsss = True
           End If
             If Sucsss Then
              For i = 1 To njob * nMachine

                 If PopCromosm(RandCrmsm1, i) <> PopCromosm(RandCrmsm2, i) Then
                  Prntunidentical = True
                  i = njob * nMachine
                 End If
               Next i
              End If
              If noftryy > memsize Then
              Sucsss = True
              Prntunidentical = True
              End If
          Loop
      For j = 1 To njob * nMachine
          Prnt1(j) = PopCromosm(RandCrmsm1, j)
         Prnt1Indx(j) = PopCromIndx(RandCrmsm1, j)
         Prnt2(j) = PopCromosm(RandCrmsm2, j)
         Prnt2Indx(j) = PopCromIndx(RandCrmsm2, j)
     Next j
End Sub

Sub PPXCrosOver()
 Randomize Time
 Dim RandCel As Integer
ReDim RndAray(1 To njob * nMachine)
j = 0
 Do While j < (njob * nMachine)

              RandCel = Int((2) * Rnd + 1)   'Generate a random number between 1 and 2

                  j = j + 1
             RndAray(j) = RandCel     'put the generated random numbers in an array

 Loop
 Dim succes1 As Boolean
 Dim succes2 As Boolean
 Dim succes3 As Boolean
 succes3 = False
 succes2 = False
 succes1 = False
  jobId = 1
  OptId = 1
 j = 0
  k = 0
 Do While j < njob * nMachine
            j = j + 1
          If RndAray(j) = 1 Then
               k = 1
                   succes1 = False
                   succes2 = False
                   succes3 = False
                  Do While k < (njob * nMachine) + 1 And succes1 = False
```

```
                    If Prnt1(k) <> 0 Then   'check if there is any Gene in the Kth location of the parent
Coromosom
                      OfspCromosm(Pnumbr, j) = Prnt1(k) 'copy the Kth Gene of the parent1 Coromosom
into the Child Cromosom

                            jobId = 1
                            Do While jobId < (njob) + 1 And succes1 = False

                              If jobId = Prnt1(k) Then  'Identify the transfered Gene
                                 succes1 = True
                              Else
                                 jobId = jobId + 1
                              End If
                            Loop

                            OptId = 1
                            Do While OptId < (nMachine) + 1 And succes2 = False

                                 If OptId = Prnt1Indx(k) Then        'Identify the transfered Gene Index
                                    succes2 = True
                                 Else
                                    OptId = OptId + 1
                                 End If

                            Loop

                            L = 1
                            Do While L < (njob * nMachine) + 1 And succes3 = False

                                    If jobId = Prnt2(L) And OptId = Prnt2Indx(L) Then        'Find the
Location of the transfred Gene in the parent2 Coromosom

                                       Prnt2(L) = 0
                                       Prnt2Indx(L) = 0
                                       succes3 = True
                                    Else
                                       L = L + 1
                                    End If
                            Loop
                          Prnt1(k) = 0
                          Prnt1Indx(k) = 0
                        Else
                         k = k + 1
                        End If

                Loop

        Else

                k = 1
                succes1 = False
                succes2 = False
                succes3 = False

                Do While k < (njob * nMachine) + 1 And succes1 = False

                    If Prnt2(k) <> 0 Then   'check if there is any Gene in the Kth location of the parent Coromosom

                      OfspCromosm(Pnumbr, j) = Prnt2(k)  'copy the Kth Gene of the parent1 Coromosom into
the Child Cromosom
                        jobId = 1
                        Do While jobId < (njob) + 1 And succes1 = False
```

```
                    If jobId = Prnt2(k) Then  'Identify the transfered Gene
                       succes1 = True
                    Else
                      jobId = jobId + 1
                    End If
                Loop

                  OptId = 1
                Do While OptId < (nMachine) + 1 And succes2 = False

                    If OptId = Prnt2Indx(k) Then  'Identify the transfered Gene Index
                         succes2 = True
                    Else
                          OptId = OptId + 1
                    End If

                Loop

                  L = 1
                Do While L < (njob * nMachine) + 1 And succes3 = False

                          If jobId = Prnt1(L) And OptId = Prnt1Indx(L) Then        'Find the
Location of the transfred Gene in the parent2 Coromosom
                              Prnt1(L) = 0
                              Prnt1Indx(L) = 0
                              succes3 = True
                          Else
                              L = L + 1
                          End If

                Loop

                    Prnt2(k) = 0
                    Prnt2Indx(k) = 0

            Else
              k = k + 1
            End If

        Loop

    End If

Loop
End Sub

Sub OfspIndx()              'Determine the Index of new Ofspring
Dim GeneLoctn As Integer
GeneLoctn = 0
ReDim jobId(1 To njob) As Integer
ReDim GeneOptIndx(1 To njob * nMachine)
ReDim OprtionReption(1 To njob) As Integer
Dim GenFnd As Boolean
GenFnd = False
For i = 1 To njob
jobId(i) = i
Next i
 k = 0
 j = 0
      Do
```

```
            k = k + 1
              j = 0
              GenFnd = False
                 Do While GenFnd = False

                    j = j + 1

                    If jobId(j) = OfspCromosm(Pnumbr, k) Then  'check if the k'th member of cromosom is job j
                       OprtionReption(j) = OprtionReption(j) + 1 'calculate the number of repetion of job j at the
coromosom
                       GeneLoctn = GeneLoctn + 1
                       GeneOptIndx(GeneLoctn) = OprtionReption(j) 'assign the number of repetion (operation
no.)of job j at cromosom
                       GenFnd = True
                    End If
                 Loop

        Loop While GeneLoctn < (njob * nMachine)

For i = 1 To (njob * nMachine)

      OfspnIndx(Pnumbr, i) = GeneOptIndx(i)
Next i
End Sub

Sub PopCromosmIndx()        'Determine the Index of new Ofspring
Dim GeneLoctn As Integer
GeneLoctn = 0
ReDim jobId(1 To njob) As Integer
ReDim GeneOptIndx(1 To njob * nMachine)
ReDim OprtionReption(1 To njob) As Integer
Dim GenFnd As Boolean
GenFnd = False
For i = 1 To njob
jobId(i) = i
Next i
 k = 0
 j = 0
         Do

            k = k + 1
              j = 0
              GenFnd = False
                 Do While GenFnd = False
                    j = j + 1
                    If jobId(j) = PopCromosm(Popmem, k) Then  'check if the k'th member of cromosom is job j
                      OprtionReption(j) = OprtionReption(j) + 1 'calculate the number of repetion of job j at the
coromosom
                       GeneLoctn = GeneLoctn + 1

                       GeneOptIndx(GeneLoctn) = OprtionReption(j) 'assign the number of repetion (operation
no.)of job j at cromosom
                       GenFnd = True
                    End If

                 Loop

        Loop While GeneLoctn < (njob * nMachine)
For i = 1 To (njob * nMachine)
      PopCromIndx(Popmem, i) = GeneOptIndx(i)
Next i
End Sub
```

```
Sub IndividulCrmosmIndx()   'Determine the Index of each individual crommosom
Dim GeneLoctn As Integer
GeneLoctn = 0
ReDim jobId(1 To njob) As Integer
ReDim GeneOptIndx(1 To njob * nMachine)
ReDim OprtionReption(1 To njob) As Integer
ReDim IndivCromosmIndx(1 To njob * nMachine)
Dim GenFnd As Boolean
GenFnd = False
For i = 1 To njob
jobId(i) = i
Next i
 k = 0
 j = 0
          Do

              k = k + 1
                j = 0
                 GenFnd = False
                    Do While GenFnd = False
                        j = j + 1
                          If jobId(j) = Cromosm(k) Then 'check if the k'th member of cromosom is job j
                            OprtionReption(j) = OprtionReption(j) + 1 'calculate the number of repetion of job j at the
coromosom
                            GeneLoctn = GeneLoctn + 1
                            GeneOptIndx(GeneLoctn) = OprtionReption(j) 'assign the number of repetion (operation
no.)of job j at cromosom
                            GenFnd = True
                          End If

                    Loop

         Loop While GeneLoctn < (njob * nMachine)
For i = 1 To (njob * nMachine)
      IndivCromosmIndx(i) = GeneOptIndx(i)
Next i
End Sub
```

E.2 Job rotation by SAMED-JR

Main module:

```
Public NofWKS As Integer
Public NofTunit As Integer
Public Wrkseqnc() As Integer
Public Skill() As Currency
Public Motivation() As Currency
Public Skill_Rem() As Currency
Public aRival() As Integer
Public Ldepart() As Integer
Public TimeToUpBoundSkil() As Integer
Public TimeToLwBoundSkil() As Integer
Public bstTDfromSPT() As Currency
Public timeTDfromSPT() As Currency
Public TimeToUpBoundMotivn() As Integer
Public TimeToLwBoundMotivn() As Integer
Public BstRTplan()
Public DMtivUpB() As Integer
'Public Kcount As Long
Public ItrnBsPlan() As Integer
Public ItrnBstTDfromSPT As Currency
Public Taow() As Currency
Public Beta() As Currency
Public Gama() As Currency
Public Teta() As Currency
Public SkilUpBound As Currency
Public SkilLwBound As Currency
Public MotivnUpBound As Currency
Public MotivnLwBound As Currency
Public Epsilon As Currency
Public Delta As Currency
Public SelctdWrkr1 As Integer
Public SelctdWrkr2 As Integer
Public NofTabu As Integer, tabuIdx As Integer, tabuFilled As Boolean
Public taboMatrix() As Integer
Public isTrap As Boolean
Public Prnt1() As Integer
Public Prnt2() As Integer
Public MaxNofty As Long
Public TempOfspCromosm() As Integer
Public PopCromosm() As Integer
Public PrevusItrnBstTDfromSPT As Currency
Public memsize As Integer
Public Popmem As Integer
Public ItrCount As Long
Public Smax As Integer
Public Smin As Integer
Public Sini As Integer
Public Alfa As Currency
Public Vmax As Integer
Public Vmin As Integer
Public Vini As Integer
Public scunt As Long
Public Jnumber1 As Integer
Public Jnumber2 As Integer
Public Ttime As Integer
Public TDfromSPT As Currency
```

```
Public TDfromSPT_Min As Currency
Public ReplictionCount As Integer
Public replicationNo As Integer
Public Makspans() As Integer
Public Solutnfound() As Integer
Public Kcnstant As Integer
Public CTemptur As Currency
Public Movcountr As Long
Public gcunt As Integer
Public Moveno1 As Currency

Sub WritData3(OutptFile3 As String)
Open OutptFile3 For Output As #1
Dim i As Integer
Dim changeLine As Boolean
changeLine = False
For i = 1 To gcunt

        Print #1, bstTDfromSPT(i)

Next i
Close #1

Exit Sub
ShErr:
  MsgBox "Error in reading from file \n Check file name and path."
End Sub

Sub WritData4(OutptFile4 As String)
On Error GoTo ShErr:
Open OutptFile4 For Output As #1
Dim i As Integer
Dim changeLine As Boolean
changeLine = False
For i = 1 To gcunt

        Print #1, timeTDfromSPT(i)

Next i
Close #1
Exit Sub
ShErr:
  MsgBox "Error in reading from file \n Check file name and path."
End Sub

Sub read_data1(DFname1 As String)
On Error GoTo ShErr:
Open DFname1 For Input As #1
Dim i As Integer
Input #1, NofWKS
ReDim Beta(1 To NofWKS) As Currency
For i = 1 To NofWKS

    Input #1, Beta(i) ' Read data to array

Next i
Close #1
Exit Sub
ShErr:
  MsgBox "Error in reading from file \n Check file name and path."

End Sub
```

```
Sub read_data2(DFname2 As String)
On Error GoTo ShErr:
Open DFname2 For Input As #1
Dim i As Integer
Input #1, NofWKS
ReDim Gama(1 To NofWKS) As Currency
For i = 1 To NofWKS

    Input #1, Gama(i) ' Read data to array

Next i
Close #1
Exit Sub
ShErr:
  MsgBox "Error in reading from file \n Check file name and path."
End Sub

Sub read_data3(DFname3 As String)
On Error GoTo ShErr:
Open DFname3 For Input As #1
Dim i As Integer
Input #1, NofWKS
ReDim Teta(1 To NofWKS) As Currency
For i = 1 To NofWKS

    Input #1, Teta(i) ' Read data to array

Next i
Close #1
Exit Sub
ShErr:
  MsgBox "Error in reading from file \n Check file name and path."
End Sub

Sub read_data4(DFname4 As String)
On Error GoTo ShErr:
Open DFname4 For Input As #1
Dim i As Integer
Input #1, NofWKS
ReDim Taow(1 To NofWKS) As Currency
For i = 1 To NofWKS

    Input #1, Taow(i) ' Read data to array

Next i
Close #1
Exit Sub
ShErr:
  MsgBox "Error in reading from file \n Check file name and path."
End Sub

Sub read_data5(DFname5 As String)
On Error GoTo ShErr:
Open DFname5 For Input As #1
Dim i As Integer
Input #1, NofWKS, NofTunit
ReDim Wrkseqnc(1 To NofWKS, 0 To NofTunit)
For i = 1 To NofWKS
  For j = 1 To NofTunit

    Input #1, Wrkseqnc(i, j) ' Read data to array
```

```
   Next j

Next i
Close #1
Exit Sub
ShErr:
  MsgBox "Error in reading from file \n Check file name and path."
End Sub

Sub Writedata5(DFname6 As String)
On Error GoTo ShErr:
Open DFname6 For Output As #1
Dim i As Integer
Dim j As Integer
Dim changeLine As Boolean
changeLine = False

Print #1,
For i = 1 To NofWKS

   For j = 1 To NofTunit

      If j = NofTunit Then
            changeLine = True
       Print #1, BstRTplan(i, j)

        changeLine = False
        Else
        Print #1, BstRTplan(i, j);          ' write data
      End If
   Next j
   changeLine = False
Next i
Close #1

Exit Sub
ShErr:
  MsgBox "Error in reading from file \n Check file name and path."

End Sub
Sub generateIinitialSolution()

Randomize Time
ReDim tempWrkseqnc(0 To NofWKS, 0 To NofTunit) As Integer
ReDim Wrkseqnc(0 To NofWKS, 0 To NofTunit) As Integer
Dim valid As Boolean
Dim AssinmValid As Boolean
Dim Rdm As Integer

      For t = 1 To NofTunit
         Do
            AssinmValid = True
           For j = 1 To NofWKS

                 Do

                    Rdm = Int((NofWKS) * Rnd + 1)
                    Wrkseqnc(j, t) = Rdm
                    k = 1
                    valid = True
                     Do
```

```
                    If Wrkseqnc(j, t) = Wrkseqnc(j - k, t) Then
                    valid = False
                    Else
                    k = k + 1
                    End If
                Loop Until k > j Or valid = False
            Loop Until k > j
              If t >= 2 Then
                If Wrkseqnc(j, t) <> Wrkseqnc(j, t - 1) Then

                    If Wrkseqnc(j, t) = Wrkseqnc(j, t - 2) Then
                      AssinmValid = False
                      j = NofWKS
                    End If
                End If
              End If


          Next j

        Loop While AssinmValid = False


      Next t

  For t = 1 To NofTunit
    For j = 1 To NofWKS
      tempWrkseqnc(j, t) = Wrkseqnc(j, t)
    Next j
  Next t
End Sub

Public Sub calculateSkill_Motivation()
ReDim Skill(1 To NofWKS, 1 To NofWKS, 0 To NofTunit)
ReDim Motivation(1 To NofWKS, 0 To NofWKS, 0 To NofTunit)
ReDim Skill_Rem(1 To NofWKS, 1 To NofWKS, 0 To NofTunit)
ReDim aRival(1 To NofWKS, 1 To NofWKS)
ReDim Ldepart(1 To NofWKS, 1 To NofWKS)
ReDim TimeToUpBoundSkil(1 To NofWKS, 1 To NofWKS)
ReDim TimeToLwBoundSkil(1 To NofWKS, 1 To NofWKS)
ReDim TimeToUpBoundMotivn(1 To NofWKS, 1 To NofWKS)
ReDim TimeToLwBoundMotivn(1 To NofWKS, 1 To NofWKS)
ReDim DMtivUpB(1 To NofWKS)
 Smax = 20
 Smin = 0
 Sini = 5
 Vmax = 20
 Vmin = 0
 Vini = 5
 Alfa = 0.5
SkilUpBound = 19.8
SkilLwBound = 0.2
MotivnUpBound = 19.8
MotivnLwBound = 0.2
Epsilon = 0.2
Delta = 0.2
For j = 1 To NofWKS

    Skill_Rem(Wrkseqnc(j, 1), j, 0) = Sini 'set the level of skill remnants for all workers and jobs to initial values at time
zero
```

```
Next j

For j = 1 To NofWKS

   Ldepart(Wrkseqnc(j, 1), j) = 0  'set the last depatuer times of all workers from workstation to zero

Next j

For j = 1 To NofWKS
   'For l = 1 To NofTunit
    Motivation(Wrkseqnc(j, 1), 0, 0) = Vini 'set the level of motivation for all workers to initial values at the first time
slice
   'Next l
Next j

For WKno = 1 To NofWKS  'selct one worker at a time from the Matrix of worker/wrkstation- time
   k = 1
   Lst = 0
   'For t = 1 To NofTunit
    WrkerID = Wrkseqnc(WKno, 1)  'pick up a worker ID
    j = WKno
         Do
              chainAssinmnt = False
              AssignBreak = False
               aRival(WrkerID, j) = k - 1  ' assign the arival time of selected worker to workstation j
                 ForwardTempT = k

                If ForwardTempT = NofTunit Then
                     chainAssinmnt = False
                Else
                    Do
                          If WrkerID = Wrkseqnc(j, ForwardTempT + 1) Then  'check if the worker is assigned to the
same workstation in consecutive period of times
                              chainAssinmnt = True
                              ForwardTempT = ForwardTempT + 1  ' if yes find the time at which the  assignment end
                          Else
                            AssignBreak = True
                          End If
                           'ForwardTempT = ForwardTempT + 1
                    Loop While AssignBreak = False And ForwardTempT < NofTunit
                End If
              If chainAssinmnt = False Then  ' if this is not a multiperiod-consecutive assignment

                    TimeToUpBoundMotivn(WrkerID, j) = (-Log(Delta / (Vmax - Motivation(WrkerID, Lst,
aRival(WrkerID, j)))))) / Taow(WrkerID) + aRival(WrkerID, j)

                    If k < TimeToUpBoundMotivn(WrkerID, j) Then

                       Motivation(WrkerID, j, k) = Vmax - (Vmax - Motivation(WrkerID, Lst, aRival(WrkerID, j))) *
Exp((-Taow(WrkerID)) * (k - aRival(WrkerID, j)))
                    Else

                        Motivation(WrkerID, j, k) = MotivnUpBound

                    End If

               If Ldepart(WrkerID, j) = 0 Then  ' check if the worker has been wrking in this station before

                  Skill_Rem(WrkerID, j, k) = Sini   'if yes, set the remnant of worker's skill equal to initial level
                  Skill(WrkerID, j, k) = Smax - (Smax - Sini) * Exp((Beta(WrkerID)) * (k - aRival(WrkerID, j)))
'calculate the skilllevel of the worker at the end of the time period k
```

```
            Else

              TimeToLwBoundSkil(WrkerID, j) = ((Log(Epsilon / Skill(WrkerID, j, Ldepart(WrkerID, j)))) /
Gama(WrkerID)) + Ldepart(WrkerID, j) ' Calculate time at which worker's skill reaches Lowerbound

              If k < TimeToLwBoundSkil(WrkerID, j) Then
                Skill_Rem(WrkerID, j, k) = Skill(WrkerID, j, Ldepart(WrkerID, j)) * Exp((Gama(WrkerID)) *
(aRival(WrkerID, j) - Ldepart(WrkerID, j))) ' otherwise, calculate the skill remnant according to the worker's skill level
when last departed from the same workstation and the ellapse time
              Else
                Skill_Rem(WrkerID, j, k) = SkilLwBound
              End If

            Skill(WrkerID, j, k) = Smax - (Smax - Skill_Rem(WrkerID, j, k)) * Exp((Beta(WrkerID)) * (k -
aRival(WrkerID, j)))

            'aRival(Wrkseqnc(j, t), j) = t - 1
            'TempT = t
          End If
        Else

            TimeToUpBoundMotivn(WrkerID, j) = ((-Log(Delta / (Vmax - Motivation(WrkerID, Lst,
aRival(WrkerID, j))))) / Taow(WrkerID)) + aRival(WrkerID, j)
            DMtivUpB(WrkerID) = (-Log(Delta / (Vmax - Vini))) / Taow(WrkerID)
            Teptt = DMtivUpB(WrkerID) + aRival(WrkerID, j)


            For i = k To ForwardTempT

              If i < TimeToUpBoundMotivn(WrkerID, j) Then

                Motivation(WrkerID, j, i) = Vmax - (Vmax - Motivation(WrkerID, Lst, aRival(WrkerID, j))) *
Exp((-Taow(WrkerID)) * (i - aRival(WrkerID, j)))
              Else

                If TimeToUpBoundMotivn(WrkerID, j) <= i And i < Teptt Then
                  Motivation(WrkerID, j, i) = MotivnUpBound
                Else


                  Motivation(WrkerID, j, i) = MotivnUpBound * Exp((-Teta(WrkerID)) * (i - Teptt))


                End If
              End If


            Next i

          'For i = k To ForwardTempT - 1

            If Ldepart(WrkerID, j) = 0 Then

              Skill_Rem(WrkerID, j, aRival(WrkerID, j)) = Sini

              TimeToUpBoundSkil(WrkerID, j) = (Log(Epsilon / (Smax - Skill_Rem(WrkerID, j, aRival(WrkerID,
j)))) / Beta(WrkerID) + aRival(WrkerID, j) 'calculate the time at which worker's skill reaches the upperbound
              For i = k To ForwardTempT  ' calculate the worker's skill at each time period of consecutive -
multiple assignment

              If i < TimeToUpBoundSkil(WrkerID, j) Then
                Skill(WrkerID, j, i) = Smax - (Smax - Sini) * Exp((Beta(WrkerID)) * (i - aRival(WrkerID, j)))
```

```
                    Else
                        Skill(WrkerID, j, i) = SkilUpBound
                    End If
                Next i

            Else

                TimeToLwBoundSkil(WrkerID, j) = ((Log(Epsilon / Skill(WrkerID, j, Ldepart(WrkerID, j)))) /
Gama(WrkerID)) + Ldepart(WrkerID, j) ' Calculate time at which worker's skill reaches Lowerbound

                    'this k may change to ARIV
                    If k < TimeToLwBoundSkil(WrkerID, j) Then ' if the time of Reassignment to station j is less
than time at which worker's skill reaches the Lowerbound
                        Skill_Rem(WrkerID, j, aRival(WrkerID, j)) = Skill(WrkerID, j, Ldepart(WrkerID, j)) *
Exp((Gama(WrkerID)) * (aRival(WrkerID, j) - Ldepart(WrkerID, j)))
                    Else
                        Skill_Rem(WrkerID, j, aRival(WrkerID, j)) = SkilLwBound ' otherwise, set the worker's skill
remnant to lowrbound
                    End If

                TimeToUpBoundSkil(WrkerID, j) = (Log(Epsilon / (Smax - Skill_Rem(WrkerID, j, aRival(WrkerID,
j)))) / Beta(WrkerID) + aRival(WrkerID, j) 'calculate the time at which worker's skill reaches the upperbound
                For i = k To ForwardTempT

                    If i < TimeToUpBoundSkil(WrkerID, j) Then
                        Skill(WrkerID, j, i) = Smax - (Smax - Skill_Rem(WrkerID, j, aRival(WrkerID, j))) *
Exp((Beta(WrkerID)) * (i - aRival(WrkerID, j)))
                    Else
                        Skill(WrkerID, j, i) = SkilUpBound
                    End If

                Next i

            End If

        End If

        Ldepart(WrkerID, j) = ForwardTempT
        k = ForwardTempT
        Lst = j
        k = k + 1

        RowNumber = 1
        NextAssmntFound = False

        If k < NofTunit + 1 Then
            Do      ' search all workstation to find the one the next assinment of the  worker
                If Wrkseqnc(RowNumber, k) = WrkerID Then
                    NextAssmntFound = True
                    j = RowNumber  ' once it is found record the workstation ID
                Else
                    RowNumber = RowNumber + 1
                End If
            Loop While NextAssmntFound = False
        End If
    Loop While k < NofTunit + 1
Next WKno

ReDim TempSumCount(0 To NofWKS * NofTunit)
SumCount = 1
TempTDfromSPT = 0
For i = 1 To NofWKS
```

```
        For j = 1 To NofWKS
            For t = 1 To NofTunit

                If Skill(i, j, t) <> 0 Then

                    TempSumCount(SumCount) = ((SkilUpBound - Skill(i, j, t)) / SkilUpBound) * Alfa +
((MotivnUpBound - Motivation(i, j, t)) / MotivnUpBound) * (1 - Alfa)
                    TempTDfromSPT = TempTDfromSPT + TempSumCount(SumCount)

                    SumCount = SumCount + 1
                End If

            Next t
        Next j
    Next i
    TDfromSPT = TempTDfromSPT

End Sub

Public Sub Generate_NeigbouringSolution()
Dim TempId As Integer
Dim inTabu As Boolean
Dim nofTry As Integer, maxTry As Integer
nofTry = 0
maxTry = 100
Randomize Time
Dim NeigbourValid As Boolean
Dim WorkersFound As Boolean
    Do

        inTabu = True
        NeigbourValid = True
        Do While inTabu   ' look until find  not in tabu

            Rdm = Int((NofWKS) * Rnd + 1) 'select a row in the workstation-worker- timeslice Matrix
              Jnumber1 = Rdm
              WorkersFound = False
            Do

              Rdm = Int((NofWKS) * Rnd + 1)
              Jnumber2 = Rdm
              If Jnumber1 <> Jnumber2 Then ' selct another row and make sure it is different from the first row
                 WorkersFound = True
              End If
            Loop While WorkersFound = False

              Rdm = Int((NofTunit) * Rnd + 1) 'selct a time slice
              Ttime = Rdm


            inTabu = isWorkerinTabu
             If inTabu Then ' If it is in tabu list then return to previous state
               nofTry = nofTry + 1
               If nofTry > maxTry Then
                 inTabu = False

               End If
             End If
        Loop

            TempId = Wrkseqnc(Jnumber1, Ttime) ' swap the workstations of the corresponding workers
            Wrkseqnc(Jnumber1, Ttime) = Wrkseqnc(Jnumber2, Ttime)
```

```
        Wrkseqnc(Jnumber2, Ttime) = TempId
        nofTry = 0

      If 1 < Ttime And Ttime < NofTunit Then 'check if selcted timeslice is neither the first nor the last one
        If Wrkseqnc(Jnumber1, Ttime + 1) = Wrkseqnc(Jnumber1, Ttime - 1) Then 'check if the new position of the
selected worker intrupt a chain assignment resulting an infeasible solution

            NeigbourValid = False
            TempId = Wrkseqnc(Jnumber1, Ttime) ' swap the workstations of the corresponding workers
            Wrkseqnc(Jnumber1, Ttime) = Wrkseqnc(Jnumber2, Ttime)
            Wrkseqnc(Jnumber2, Ttime) = TempId

      Else
        If Wrkseqnc(Jnumber2, Ttime + 1) = Wrkseqnc(Jnumber2, Ttime - 1) Then
          NeigbourValid = False
          TempId = Wrkseqnc(Jnumber1, Ttime) ' swap the workstations of the corresponding workers
          Wrkseqnc(Jnumber1, Ttime) = Wrkseqnc(Jnumber2, Ttime)
          Wrkseqnc(Jnumber2, Ttime) = TempId
        End If
      End If
    End If
  If NeigbourValid = True Then

    For mcount = 1 To 2  'pick up one of the workers whose workstation is going to be changed
      If mcount = 1 Then
        j = Jnumber1
      Else
        j = Jnumber2
      End If

      If 1 < Ttime And Ttime <= NofTunit - 2 Then '  if the selected time slice is greater than one and less than and
equal to the total number of timeslices minus one

        If Wrkseqnc(j, Ttime) <> Wrkseqnc(j, Ttime - 1) Then ' if the selected worker at workerstation "j" differ
from the one in the previous timeslice

          If Wrkseqnc(j, Ttime) = Wrkseqnc(j, Ttime - 2) Then ' But it is the same worker at  the "current
timeslice minus two"

            NeigbourValid = False                ' then this is an infeasible solution
            TempId = Wrkseqnc(Jnumber1, Ttime) ' swap the workstations of the corresponding workers
            Wrkseqnc(Jnumber1, Ttime) = Wrkseqnc(Jnumber2, Ttime)
            Wrkseqnc(Jnumber2, Ttime) = TempId

          Else

            If Wrkseqnc(j, Ttime) <> Wrkseqnc(j, Ttime + 1) Then 'otherwise check if the workers IDs
operating in the same workstation "j" are diffrent (at the selcted time slice and at the next timeslice)

              If Wrkseqnc(j, Ttime) = Wrkseqnc(j, Ttime + 2) Then ' then check if they are identical at the
selected timeslice and at the one two unit after

                NeigbourValid = False  'if yes the solution is infeasible
                TempId = Wrkseqnc(Jnumber1, Ttime) ' swap the workstations of the corresponding
workers
                Wrkseqnc(Jnumber1, Ttime) = Wrkseqnc(Jnumber2, Ttime)
                Wrkseqnc(Jnumber2, Ttime) = TempId

              End If

            End If
          End If
```

```
            Else

                If Wrkseqnc(j, Ttime) <> Wrkseqnc(j, Ttime + 1) Then

                    If Wrkseqnc(j, Ttime) = Wrkseqnc(j, Ttime + 2) Then

                      NeigbourValid = False
                      TempId = Wrkseqnc(Jnumber1, Ttime) ' swap the workstations of the corresponding
workers
                       Wrkseqnc(Jnumber1, Ttime) = Wrkseqnc(Jnumber2, Ttime)
                       Wrkseqnc(Jnumber2, Ttime) = TempId

                    End If

                End If
            End If
        Else

            If Ttime = 1 Then

                If Wrkseqnc(j, Ttime) <> Wrkseqnc(j, Ttime + 1) Then

                    If Wrkseqnc(j, Ttime) = Wrkseqnc(j, Ttime + 2) Then

                      NeigbourValid = False
                      TempId = Wrkseqnc(Jnumber1, Ttime) ' swap the workstations of the corresponding workers
                       Wrkseqnc(Jnumber1, Ttime) = Wrkseqnc(Jnumber2, Ttime)
                       Wrkseqnc(Jnumber2, Ttime) = TempId

                    End If

                End If

            Else

                If Ttime = NofTunit Then
                  If Wrkseqnc(j, Ttime) <> Wrkseqnc(j, Ttime - 1) Then

                    If Wrkseqnc(j, Ttime) = Wrkseqnc(j, Ttime - 2) Then

                      NeigbourValid = False
                      TempId = Wrkseqnc(Jnumber1, Ttime) ' swap the workstations of the corresponding workers
                       Wrkseqnc(Jnumber1, Ttime) = Wrkseqnc(Jnumber2, Ttime)
                       Wrkseqnc(Jnumber2, Ttime) = TempId

                    End If


                  End If
                Else
                  If Wrkseqnc(j, Ttime) <> Wrkseqnc(j, Ttime - 1) Then

                    If Wrkseqnc(j, Ttime) = Wrkseqnc(j, Ttime - 2) Then

                      NeigbourValid = False
                      TempId = Wrkseqnc(Jnumber1, Ttime) ' swap the workstations of the corresponding workers
                       Wrkseqnc(Jnumber1, Ttime) = Wrkseqnc(Jnumber2, Ttime)
                       Wrkseqnc(Jnumber2, Ttime) = TempId
                    Else
                      If Wrkseqnc(j, Ttime + 1) = Wrkseqnc(j, Ttime - 1) Then
```

```
                        NeigbourValid = False
                        TempId = Wrkseqnc(Jnumber1, Ttime) ' swap the workstations of the corresponding workers
                        Wrkseqnc(Jnumber1, Ttime) = Wrkseqnc(Jnumber2, Ttime)
                        Wrkseqnc(Jnumber2, Ttime) = TempId
                    End If
                End If
            End If
        End If
    End If
End If

If NeigbourValid = False Then
    mcount = mcount + 1
End If

Next mcount
End If

Loop While NeigbourValid = False

End Sub

Public Sub ComputeNewSkill_Motivation()

For j = 1 To NofWKS

Skill_Rem(Wrkseqnc(Jnumber1, Ttime), j, 0) = Sini 'set the level of skill remnants for the two workers and jobs to initial values at time zero
Skill_Rem(Wrkseqnc(Jnumber2, Ttime), j, 0) = Sini

Next j

For j = 1 To NofWKS

Ldepart(Wrkseqnc(Jnumber1, Ttime), j) = 0 'set the last depatuer times of the two workers from workstation to zero
Ldepart(Wrkseqnc(Jnumber2, Ttime), j) = 0
Next j

Motivation(Wrkseqnc(Jnumber1, Ttime), 0, 0) = Vini 'set the level of motivation for the two workers to initial values at the first time slice
Motivation(Wrkseqnc(Jnumber2, Ttime), 0, 0) = Vini

Dim jcunt As Integer
Dim tcunt As Integer
For jcunt = 1 To NofWKS
  For tcunt = 1 To NofTunit
      Motivation(Wrkseqnc(Jnumber1, Ttime), jcunt, tcunt) = 0
      Motivation(Wrkseqnc(Jnumber2, Ttime), jcunt, tcunt) = 0
      Skill(Wrkseqnc(Jnumber1, Ttime), jcunt, tcunt) = 0
      Skill(Wrkseqnc(Jnumber2, Ttime), jcunt, tcunt) = 0
  Next tcunt
Next jcunt
For WKno = 1 To 2
  Lst = 0
  If WKno = 1 Then
    WrkerID = Wrkseqnc(Jnumber1, Ttime) 'pick up a worker ID
    n = 1
    wrker_WorkstnFound = False
    Do
      If Wrkseqnc(n, 1) = WrkerID Then
        wrker_WorkstnFound = True
          j = n
```

```
            k = 1
        End If
        n = n + 1
    Loop While wrker_WorkstnFound = False

Else

    WrkerID = Wrkseqnc(Jnumber2, Ttime)  'pick up a worker ID
    n = 1
    wrker_WorkstnFound = False
    Do
      If Wrkseqnc(n, 1) = WrkerID Then
         wrker_WorkstnFound = True
           j = n
           k = 1
      End If
       n = n + 1
    Loop While wrker_WorkstnFound = False

End If
     Do
          chainAssinmnt = False
          AssignBreak = False
          aRival(WrkerID, j) = k - 1  ' assign the arival time of selected worker to workstation j
            ForwardTempT = k

            If ForwardTempT = NofTunit Then
                chainAssinmnt = False
            Else
                Do
                    If WrkerID = Wrkseqnc(j, ForwardTempT + 1) Then  'check if the worker is assigned to the
same workstation in consecutive period of times
                        chainAssinmnt = True
                        ForwardTempT = ForwardTempT + 1  ' if yes find the time at which the  assignment end
                    Else
                      AssignBreak = True
                    End If

                Loop While AssignBreak = False And ForwardTempT < NofTunit
            End If
          If chainAssinmnt = False Then  ' if this is not a multiperiod-consecutive assignment

                TimeToUpBoundMotivn(WrkerID, j) = (-Log(Delta / (Vmax - Motivation(WrkerID, Lst,
aRival(WrkerID, j)))))) / Taow(WrkerID) + aRival(WrkerID, j)

                If k < TimeToUpBoundMotivn(WrkerID, j) Then

                   Motivation(WrkerID, j, k) = Vmax - (Vmax - Motivation(WrkerID, Lst, aRival(WrkerID, j))) *
Exp((-Taow(WrkerID)) * (k - aRival(WrkerID, j)))
                Else

                    Motivation(WrkerID, j, k) = MotivnUpBound

                End If

            If Ldepart(WrkerID, j) = 0 Then  ' check if the worker has been wrking in this station before

              Skill_Rem(WrkerID, j, k) = Sini   'if yes, set the remnant of worker's skill equal to initial level
              Skill(WrkerID, j, k) = Smax - (Smax - Sini) * Exp((Beta(WrkerID)) * (k - aRival(WrkerID, j)))
'calculate the skilllevel of the worker at the end of the time period k

            Else
```

```
            TimeToLwBoundSkil(WrkerID, j) = ((Log(Epsilon / Skill(WrkerID, j, Ldepart(WrkerID, j)))) /
Gama(WrkerID)) + Ldepart(WrkerID, j) ' Calculate time at which worker's skill reaches Lowerbound

            If k < TimeToLwBoundSkil(WrkerID, j) Then
              Skill_Rem(WrkerID, j, k) = Skill(WrkerID, j, Ldepart(WrkerID, j)) * Exp((Gama(WrkerID)) *
(aRival(WrkerID, j) - Ldepart(WrkerID, j))) ' otherwise, calculate the skill remnant according to the worker's skill level
when last departed from the same workstation and the ellapse time
            Else
              Skill_Rem(WrkerID, j, k) = SkilLwBound
            End If

          Skill(WrkerID, j, k) = Smax - (Smax - Skill_Rem(WrkerID, j, k)) * Exp((Beta(WrkerID)) * (k -
aRival(WrkerID, j)))

        End If
      Else

            TimeToUpBoundMotivn(WrkerID, j) = ((-Log(Delta / (Vmax - Motivation(WrkerID, Lst,
aRival(WrkerID, j))))) / Taow(WrkerID)) + aRival(WrkerID, j)
            DMtivUpB(WrkerID) = (-Log(Delta / (Vmax - Vini))) / Taow(WrkerID)
            Teptt = DMtivUpB(WrkerID) + aRival(WrkerID, j)


            For i = k To ForwardTempT

              If i < TimeToUpBoundMotivn(WrkerID, j) Then

                Motivation(WrkerID, j, i) = Vmax - (Vmax - Motivation(WrkerID, Lst, aRival(WrkerID, j))) *
Exp((-Taow(WrkerID)) * (i - aRival(WrkerID, j)))
              Else

                If TimeToUpBoundMotivn(WrkerID, j) <= i And i < Teptt Then
                  Motivation(WrkerID, j, i) = MotivnUpBound
                Else

                  Motivation(WrkerID, j, i) = MotivnUpBound * Exp((-Teta(WrkerID)) * (i - Teptt))


                End If
              End If

            Next i

            If Ldepart(WrkerID, j) = 0 Then

              Skill_Rem(WrkerID, j, aRival(WrkerID, j)) = Sini

              TimeToUpBoundSkil(WrkerID, j) = (Log(Epsilon / (Smax - Skill_Rem(WrkerID, j, aRival(WrkerID,
j))))) / Beta(WrkerID) + aRival(WrkerID, j) 'calculate the time at which worker's skill reaches the upperbound
              For i = k To ForwardTempT  ' calculate the worker's skill at each time period of consecutive -
multiple assignment

                If i < TimeToUpBoundSkil(WrkerID, j) Then
                  Skill(WrkerID, j, i) = Smax - (Smax - Sini) * Exp((Beta(WrkerID)) * (i - aRival(WrkerID, j)))
                Else
                  Skill(WrkerID, j, i) = SkilUpBound
                End If
              Next i

            Else
```

```
                    TimeToLwBoundSkil(WrkerID, j) = ((Log(Epsilon / Skill(WrkerID, j, Ldepart(WrkerID, j)))) /
Gama(WrkerID)) + Ldepart(WrkerID, j) ' Calculate time at which worker's skill reaches Lowerbound

                        'this k may change to ARIV
                        If k < TimeToLwBoundSkil(WrkerID, j) Then ' if the time of Reassignment to station j is less
than time at which worker's skill reaches the Lowerbound
                          Skill_Rem(WrkerID, j, aRival(WrkerID, j)) = Skill(WrkerID, j, Ldepart(WrkerID, j)) *
Exp((Gama(WrkerID)) * (aRival(WrkerID, j) - Ldepart(WrkerID, j)))
                        Else
                            Skill_Rem(WrkerID, j, aRival(WrkerID, j)) = SkilLwBound ' otherwise, set the worker's skill
remnant to lowrbound
                        End If

                    TimeToUpBoundSkil(WrkerID, j) = (Log(Epsilon / (Smax - Skill_Rem(WrkerID, j, aRival(WrkerID,
j)))) / Beta(WrkerID) + aRival(WrkerID, j) 'calculate the time at which worker's skill reaches the upperbound
                  For i = k To ForwardTempT

                      If i < TimeToUpBoundSkil(WrkerID, j) Then
                         Skill(WrkerID, j, i) = Smax - (Smax - Skill_Rem(WrkerID, j, aRival(WrkerID, j))) *
Exp((Beta(WrkerID)) * (i - aRival(WrkerID, j)))
                      Else
                         Skill(WrkerID, j, i) = SkilUpBound
                      End If

                  Next i

                End If

            End If

              Ldepart(WrkerID, j) = ForwardTempT
              k = ForwardTempT
               Lst = j
              k = k + 1
                RowNumber = 1
                NextAssmntFound = False

                If k < NofTunit + 1 Then
                     Do      ' search all workstation to find the one the next assinment of the  worker
                       If Wrkseqnc(RowNumber, k) = WrkerID Then
                          NextAssmntFound = True
                          j = RowNumber  ' once it is found record the workstation ID
                       Else
                          RowNumber = RowNumber + 1
                       End If
                     Loop While NextAssmntFound = False
                End If
        Loop While k < NofTunit + 1
 Next WKno

   ReDim TempSumCount(0 To NofWKS * NofTunit)
    SumCount = 1
    TempTDfromSPT = 0
   For i = 1 To NofWKS
        For j = 1 To NofWKS
             For t = 1 To NofTunit

                  If Skill(i, j, t) <> 0 Then

                    TempSumCount(SumCount) = ((SkilUpBound - Skill(i, j, t)) / SkilUpBound) * Alfa +
((MotivnUpBound - Motivation(i, j, t)) / MotivnUpBound) * (1 - Alfa)
                    TempTDfromSPT = TempTDfromSPT + TempSumCount(SumCount)
```

```
                SumCount = SumCount + 1
            End If

        Next t
    Next j
  Next i
  TDfromSPT = TempTDfromSPT

End Sub

Sub ResetNeighbour()
Dim TempId As Integer
TempId = Wrkseqnc(Jnumber1, Ttime) ' swap the workstations of the corresponding workers
Wrkseqnc(Jnumber1, Ttime) = Wrkseqnc(Jnumber2, Ttime)
Wrkseqnc(Jnumber2, Ttime) = TempId

End Sub

Sub nextMove()
'Dim ConlPmtr As Double
Dim CostDiffrenc As Integer
Dim PrbOfacptance As Double
Dim F As Currency
Dim MRdn As Currency
Dim Success As Boolean

If tabuFilled = True And isTrap = False Then
   TDfromSPT = ItrnBstTDfromSPT
End If
tabuFilled = False
Cnbhd = TDfromSPT
neibourNO = 0
Dim ccCoTime As Integer
isTrap = False

Do

    Generate_NeigbouringSolution

    ComputeNewSkill_Motivation

  CostDiffrenc = TDfromSPT - Cnbhd
  F = CostDiffrenc / CTemptur
  PrbOfacptance = Exp(-F)
  If PrbOfacptance >= 1 Then
     Success = True
  Else
     MRdn = Rnd
     If MRdn < PrbOfacptance Then
     Success = True
     Else

        ResetNeighbour
          ccCoTime = TDfromSPT
          TDfromSPT = Cnbhd

           Nofty = Nofty + 1

              If Nofty > MaxNofty Then
                 Success = True
                  ResetNeighbour
```

```
                TDfromSPT = ccCoTime
                isTrap = True
              End If

    End If
  End If
Loop Until Success

addWorkersToTabu
Nofty = 0

      If tabuFilled = True Then

        If isTrap = False Then

            If TDfromSPT <= ItrnBstTDfromSPT Then
               ItrnBstTDfromSPT = TDfromSPT
               ItrnBestSchdul
               Bstsultion

            Else
               Bstsultion

            End If

            If ItrCount = 0 Then
                Popmem = Popmem + 1

                If Popmem = memsize + 1 Then
                  ItrCount = ItrCount + 1
                  'prvCoTmin = CoTmin
                  PrevusItrnBstTDfromSPT = ItrnBstTDfromSPT
                  Popmem = 1
                End If


            Else

                  If PrevusItrnBstTDfromSPT = ItrnBstTDfromSPT Then

                     For icunt = 1 To NofWKS
                       For jcunt = 1 To NofTunit

                          PopCromosm(Popmem, icunt, jcunt) = Wrkseqnc(icunt, jcunt)
                       Next jcunt
                     Next icunt

                      Popmem = Popmem + 1

                      If Popmem = memsize + 1 Then
                         ItrCount = ItrCount + 1

                          Popmem = 1
                          GAComponent

                          PrevusItrnBstTDfromSPT = ItrnBstTDfromSPT

                      End If
                  Else

                    Popmem = Popmem + 1
                    If Popmem = memsize + 1 Then
```

```
                    ItrCount = ItrCount + 1
                    PrevusItrnBstTDfromSPT = ItrnBstTDfromSPT
                    Popmem = 1

                End If

            End If


        End If


    Else

        If ItrCount = 0 Then
            Popmem = Popmem + 1

            If Popmem = memsize + 1 Then
                ItrCount = ItrCount + 1
                PrevusItrnBstTDfromSPT = ItrnBstTDfromSPT
                Popmem = 1
            End If

        Else
            If PrevusItrnBstTDfromSPT = ItrnBstTDfromSPT Then

                For icunt = 1 To NofWKS
                    For jcunt = 1 To NofTunit
                      opCromosm(Popmem, icunt, jcunt) = Wrkseqnc(icunt, jcunt)
                    Next jcunt
                Next icunt

                 Popmem = Popmem + 1

                If Popmem = memsize + 1 Then
                    ItrCount = ItrCount + 1

                     Popmem = 1
                     GAComponent

                     PrevusItrnBstTDfromSPT = ItrnBstTDfromSPT

                End If
            Else

                Popmem = Popmem + 1
                If Popmem = memsize + 1 Then
                    ItrCount = ItrCount + 1
                    PrevusItrnBstTDfromSPT = ItrnBstTDfromSPT
                    Popmem = 1

                End If

            End If

        End If
    End If

Else

        If TDfromSPT <= ItrnBstTDfromSPT Then
```

```
                    ItrnBstTDfromSPT = TDfromSPT
                    ItrnBestSchdul

                End If

        End If

End Sub

Sub SimulationSearch(maxMove As Integer)
   gcunt = 1
   ReplictionCount = 0
   ReDim Makspans(0 To replicationNo)
   ReDim Solutnfound(0 To replicationNo)
  ReDim bstTDfromSPT(1 To 2000)
  ReDim timeTDfromSPT(1 To 2000)

   Do While ReplictionCount < replicationNo

        Dim SearchTime As Double, Start As Double, Finish As Double, MinTime As Double
        Start = Timer
        TStart = Timer
        SearchTime = Form2.MaxMov.Text
        Tsearchtime = Form2.MaxMov.Text
        memsize = Form2.msize.Text
        MinTime = Start
        TDfromSPT_Min = TDfromSPT
          ItrCount = 0
           Popmem = 0

       Movcountr = 1
       MaxNofty = 1000
       ReDim PopCromosm(1 To memsize, 1 To NofWKS, 1 To NofTunit)
       ItrnBstTDfromSPT = TDfromSPT
       ReDim ItrnBsPlan(1 To NofWKS, 1 To NofTunit)

       For i = 1 To NofWKS
          For j = 1 To NofTunit
            ItrnBsPlan(i, j) = Wrkseqnc(i, j)
          Next j
       Next i
       Gamaa = 2
       CTemptur = 10
       Kcnstant = 10
       Movcountr = 1
       Do While Timer < Start + SearchTime

            nextMove

          CTemptur = Kcnstant / (1 + Gamaa * Log(Movcountr))
          If CTemptur < 0.0001 Then
             CTemptur = 0.0001
          End If

            If TDfromSPT < TDfromSPT_Min Then
               TDfromSPT_Min = TDfromSPT

               BestRtplan

               Moveno1 = Timer - Start

                bstTDfromSPT(gcunt) = TDfromSPT
```

```
                timeTDfromSPT(gcunt) = Moveno1
                gcunt = gcunt + 1

              End If
              If (Movcountr Mod 10) = 0 Then
                Form2.caLabel.Caption = "Calculating...." & TDfromSPT_Min
                Form2.caLabel.Refresh
              End If
            Movcountr = Movcountr + 1
        Loop

        ReplictionCount = ReplictionCount + 1

        Makspans(ReplictionCount) = TDfromSPT_Min
       Solutnfound(ReplictionCount) = Moveno1
  Loop
End Sub

Public Sub BestRtplan()
 ReDim BstRTplan(1 To NofWKS, 1 To NofTunit)
For i = 1 To NofWKS
   For j = 1 To NofTunit
     BstRTplan(i, j) = Wrkseqnc(i, j)
   Next j
Next i
End Sub
Sub addWorkersToTabu()

taboMatrix(Jnumber1, Ttime) = Wrkseqnc(Jnumber1, Ttime)
taboMatrix(Jnumber2, Ttime) = Wrkseqnc(Jnumber2, Ttime)
  If tabuIdx = NofTabu Then

    ReDim taboMatrix(1 To NofWKS, 1 To NofTunit) As Integer

    tabuIdx = 1
    tabuFilled = True ' changes first time that tabu has been filled
  Else
    tabuIdx = tabuIdx + 1
  End If
End Sub

Function isWorkerinTabu() As Boolean
Dim FoundInTabu As Boolean ', Equal As Boolean
FoundInTabu = False
If taboMatrix(Jnumber1, Ttime) = Wrkseqnc(Jnumber1, Ttime) Or taboMatrix(Jnumber2, Ttime) =
Wrkseqnc(Jnumber2, Ttime) Then
FoundInTabu = True
End If
isWorkerinTabu = FoundInTabu
End Function

Public Sub ItrnBestSchdul()
ReDim ItrnBsPlan(1 To NofWKS, 1 To NofTunit)
For i = 1 To NofWKS
   For j = 1 To NofTunit
     ItrnBsPlan(i, j) = Wrkseqnc(i, j)
   Next j
Next i
End Sub
Public Sub Bstsultion()
   For i = 1 To NofWKS
      For j = 1 To NofTunit
```

```
        Wrkseqnc(i, j) = ItrnBsPlan(i, j)
      Next j
   Next i
End Sub
Sub ParntSelection()

 Dim RandCrmsm1 As Integer
 Dim RandCrmsm2 As Integer
 Dim Sucsss As Boolean
 ReDim Prnt1(1 To NofWKS, 1 To NofTunit)
 ReDim Prnt2(1 To NofWKS, 1 To NofTunit)
 Randomize Time
 RandCrmsm1 = Int((memsize) * Rnd + 1) 'Select Parent 1 and Parent 2 randomly from the Population
 Sucsss = False
        Do While Sucsss = False

          RandCrmsm2 = Int((memsize) * Rnd + 1)

          If RandCrmsm1 <> RandCrmsm2 Then
              Sucsss = True

          End If
        Loop

   For i = 1 To NofWKS
     For j = 1 To NofTunit

       Prnt1(i, j) = PopCromosm(RandCrmsm1, i, j)

       Prnt2(i, j) = PopCromosm(RandCrmsm2, i, j)
     Next j

   Next i

End Sub

Sub CrossOver_Operator()
Randomize Time

Do
     Operation = True
     ReDim TempOfspCromosm(1 To 2, 1 To NofWKS, 1 To NofTunit)

      RandColumn = Int((NofTunit - 1) * Rnd + 1) 'generate a random number between 1 and (NoTunit-1) to select a
column

     For i = 1 To NofWKS
       For j = 1 To RandColumn
         TempOfspCromosm(1, i, j) = Prnt1(i, j) 'copy the columns before the selected colmen from each parent to a
temporayr corromosom
         TempOfspCromosm(2, i, j) = Prnt2(i, j)
       Next j
     Next i

     For i = 1 To NofWKS
       For j = RandColumn + 1 To NofTunit
         TempOfspCromosm(1, i, j) = Prnt2(i, j) 'switch the second part of the remeining columns in the temporary
chromosoms
         TempOfspCromosm(2, i, j) = Prnt1(i, j)
       Next j
     Next i
```

```
RandRow = Int((NofWKS - 1) * Rnd + 1) ' select a row randomelybetween 1 and NofWKS-1

    If RandRow <> 1 Then ' if the selected row is not the first row of the matrix then (if the selected row is 1
no need to check)

        For i = RandRow To NofWKS 'copy the elements on the selected column below the selected row
from parent2 to tem chromosem1 and do the same for paren2

            TempOfspCromosm(1, i, RandColumn) = Prnt2(i, RandColumn)
            TempOfspCromosm(2, i, RandColumn) = Prnt1(i, RandColumn)

        Next i


        For j = RandRow To NofWKS
            k = 1
            Do    'check if there is a repetetive number in the selected column


              If TempOfspCromosm(1, j, RandColumn) = TempOfspCromosm(1, j - k, RandColumn)
Then

                For i = RandRow To NofWKS

                    TempOfspCromosm(1, i, RandColumn) = Prnt1(i, RandColumn)

                Next i

              End If

              If TempOfspCromosm(2, j, RandColumn) = TempOfspCromosm(2, j - k, RandColumn)
Then

                For i = RandRow To NofWKS

                    TempOfspCromosm(2, i, RandColumn) = Prnt2(i, RandColumn)

                Next i

              End If


              k = k + 1

            Loop While k < j

        Next j

    End If


    k = 1

    Do    'check if the second constraint has been violated

      If RandColumn > 2 And RandColumn <= NofTunit - 2 Then

        For j = 1 To NofWKS
          If TempOfspCromosm(k, j, RandColumn) <> TempOfspCromosm(k, j, RandColumn - 1) Then

            If TempOfspCromosm(k, j, RandColumn) = TempOfspCromosm(k, j, RandColumn - 2) Then
```

```
                    Operation = False

                End If
            End If
        Next j

    If Operation = True Then
        For j = 1 To NofWKS
            If TempOfspCromosm(k, j, RandColumn) <> TempOfspCromosm(k, j, RandColumn + 1) Then

                If TempOfspCromosm(k, j, RandColumn + 1) = TempOfspCromosm(k, j, RandColumn - 1)
Then
                    Operation = False


                Else
                    If TempOfspCromosm(k, j, RandColumn) = TempOfspCromosm(k, j, RandColumn + 2)
Then
                        Operation = False
                    End If
                End If
            End If
        Next j

    End If

    If Operation = True Then
        For j = 1 To NofWKS
            If TempOfspCromosm(k, j, RandColumn + 1) <> TempOfspCromosm(k, j, RandColumn + 2)
Then

                If TempOfspCromosm(k, j, RandColumn) = TempOfspCromosm(k, j, RandColumn + 2) Then
                    Operation = False

                End If
            End If
        Next j
    End If

Else
    If RandColumn = 2 Then

        For j = 1 To NofWKS
            If TempOfspCromosm(k, j, RandColumn) <> TempOfspCromosm(k, j, RandColumn - 1)
Then

                If TempOfspCromosm(k, j, RandColumn + 1) = TempOfspCromosm(k, j, RandColumn -
1) Then
                    Operation = False

                End If
            End If
        Next j

        If Operation = True Then
            For j = 1 To NofWKS
                If TempOfspCromosm(k, j, RandColumn) <> TempOfspCromosm(k, j, RandColumn + 1)
Then

                If TempOfspCromosm(k, j, RandColumn) = TempOfspCromosm(k, j, RandColumn + 2)
Then
                    Operation = False
```

```
                    End If
                End If
            Next j
            End If

        Else

            If RandColumn = NofTunit - 1 Then

                For j = 1 To NofWKS
                    If TempOfspCromosm(k, j, RandColumn) <> TempOfspCromosm(k, j, RandColumn -
1) Then

                        If TempOfspCromosm(k, j, RandColumn) = TempOfspCromosm(k, j, RandColumn
- 2) Then
                            Operation = False

                        End If
                    End If
                Next j

                If Operation = True Then
                    For j = 1 To NofWKS
                        If TempOfspCromosm(k, j, RandColumn) <> TempOfspCromosm(k, j, RandColumn +
1) Then

                            If TempOfspCromosm(k, j, RandColumn + 1) = TempOfspCromosm(k, j,
RandColumn - 1) Then
                                Operation = False

                            End If
                        End If
                    Next j
                End If

            Else
                If RandColumn = 1 Then

                    For j = 1 To NofWKS
                        If TempOfspCromosm(k, j, RandColumn) <> TempOfspCromosm(k, j, RandColumn + 1)
Then

                            If TempOfspCromosm(k, j, RandColumn) = TempOfspCromosm(k, j, RandColumn +
2) Then
                                Operation = False
                            End If
                        End If
                    Next j
                End If

            End If
        End If
    End If

    k = k + 1

Loop While k <= 2 And Operation = True


    If Operation = False Then
      ParntSelection
    End If
```

```
Loop While Operation = False

End Sub

Public Sub GAComponent()
Dim Bingoo As Boolean
  Dim RanndomN As Currency
  Dim RndDD As Integer
  Pnumbr = 1
  Dim Noftrry As Integer
  ReDim OfspCromosm(1 To memsize, 1 To NofWKS, 1 To NofTunit)
  ReDim OfsTDfromSPT(1 To memsize)
  Do While Pnumbr < memsize + 1
      ParntSelection

      CrossOver_Operator

     ReDim TempOfsTDfromSPT(1 To 2)
      For k = 1 To 2

        For i = 1 To NofWKS
          For j = 1 To NofTunit

          Wrkseqnc(i, j) = TempOfspCromosm(k, i, j)

          Next j
        Next i
       calculateSkill_Motivation
       TempOfsTDfromSPT(k) = TDfromSPT
      Next k

        If TempOfsTDfromSPT(1) < TempOfsTDfromSPT(2) Then

          For i = 1 To NofWKS
            For j = 1 To NofTunit

            OfspCromosm(Pnumbr, i, j) = TempOfspCromosm(1, i, j)

            Next j
          Next i

          OfsTDfromSPT(Pnumbr) = TempOfsTDfromSPT(1)
           TDfromSPT = TempOfsTDfromSPT(1)
       Else

          For i = 1 To NofWKS
            For j = 1 To NofTunit

            OfspCromosm(Pnumbr, i, j) = TempOfspCromosm(2, i, j)

            Next j
          Next i

          OfsTDfromSPT(Pnumbr) = TempOfsTDfromSPT(2)
          TDfromSPT = TempOfsTDfromSPT(2)
      End If

        If TDfromSPT < ItrnBstTDfromSPT Then
          ItrnBstTDfromSPT = TDfromSPT
          ItrnBestSchdul
        End If
```

```
        Pnumbr = Pnumbr + 1

  Loop

If ItrnBstTDfromSPT < TDfromSPT_Min Then

      Bstsultion
      TDfromSPT = ItrnBstTDfromSPT
Else

            Randomize Time
             RanndomN = Rnd
            If RanndomN < (0.5 * (1 - (CTemptur / Kcnstant))) Then
                CTemptur = Kcnstant
                Movcountr = 1
                Noftrry = 1
                Bingoo = False

              Do While Noftrry < (memsize + 1) And Bingoo = False

                    If ItrnBstTDfromSPT <> OfsTDfromSPT(Noftrry) Then
                      Bingoo = True

                      For i = 1 To NofWKS
                          For j = 1 To NofTunit

                            Wrkseqnc(i, j) = OfspCromosm(Noftrry, i, j)
                          Next j
                        Next i

                          ItrnBstTDfromSPT = OfsTDfromSPT(Noftrry)

                          TDfromSPT = OfsTDfromSPT(Noftrry)
                          ItrCount = 0
                    End If

                    Noftrry = Noftrry + 1
               Loop

            Else
              Bstsultion

              TDfromSPT = ItrnBstTDfromSPT
            End If

 End If

      If (scunt Mod 1) = 0 Then
        Form2.Ca2labl.Caption = "Calculating...." & ItrnBstTDfromSPT
        Form2.Ca2labl.Refresh
      End If

       scunt = scunt + 1

End Sub
```

.....

Form1 (Mainform.frm)

```
Private Sub DrawCmd_Click()
```

```
NofWKS = Form1.Workstation.Text
NofTunit = Form1.PrHorizon.Text
End Sub

Private Sub Readdata1Cmd_Click()
 read_data1 DFname1.Text
 read_data2 DFname2.Text
 read_data3 DFname3.Text
 read_data4 DFname4.Text

End Sub
 Private Sub NextForm_click()
      generatelinitialSolution
      Form2.Show
      Form2.drawTable
 End Sub
Private Sub StartFrmFile_click()
read_data5 DFname5.Text
Form2.Show
Form2.drawTable
End Sub

Private Sub Continue_click()
If Option1.Value = True Then
read_data5 DFname5.Text
Form2.Show
Form2.drawTable
Else
   If Option2.Value = True Then
    generatelinitialSolution
      Form2.Show
      Form2.drawTable
   End If
End If
End Sub
```

.....

Form2(VeiwDForm.frm)

```
Dim Tlns() As Line
Public Sub drawTable()
On Error GoTo ShErr:
Dim i As Integer, j As Integer
Tabpic.Scale (-1, (NofWKS + 1))-((NofTunit + 1), -1)
Tabpic.Cls
Tabpic.Line (0, 0)-(0, NofWKS)
Tabpic.Line (0, 0)-(NofTunit, 0)
FillColor = 2
For i = 1 To NofWKS
  Tabpic.Line (0, i)-(NofTunit, i)
Next i
For i = 1 To NofWKS
   Tabpic.Line (i, 0)-(i, NofWKS)
Next i
For i = 1 To NofWKS
   For j = 1 To NofTunit
      Tabpic.CurrentY = (NofWKS - i + 0.75) - 0.1
      Tabpic.CurrentX = j - 0.8
      Tabpic.Print Wrkseqnc(i, j)
   Next j
Next i
```

```
Tabpic.FontBold = True
For i = 1 To NofWKS
  Tabpic.CurrentX = -0.3
  Tabpic.CurrentY = (NofWKS - i + 0.75) - 0.1
  Tabpic.Print " " & i
Next i
Tabpic.CurrentY = NofWKS + 0.9
Tabpic.CurrentX = (NofTunit / 2) - 3
Tabpic.Print "Worker alocation over the production horizon:"
Tabpic.CurrentY = NofWKS
For i = 1 To NofTunit
  Tabpic.CurrentY = NofTunit + 0.4
  Tabpic.CurrentX = i - 0.8
  Tabpic.Print "t" & i
Next i
Tabpic.FontBold = False
Exit Sub
ShErr:
  MsgBox "Error!" & Err.Description
End Sub

Public Sub drwTblSchdl()
On Error GoTo ShErr:
Dim i As Integer, j As Integer
Tabpic.Scale (-1, (NofWKS + 1))-((NofTunit + 1), -1)
Tabpic.Cls
Tabpic.Line (0, 0)-(0, NofWKS)
Tabpic.Line (0, 0)-(NofTunit, 0)
FillColor = 2
For i = 1 To NofWKS
  Tabpic.Line (0, i)-(NofTunit, i)
Next i
For i = 1 To NofWKS
   Tabpic.Line (i, 0)-(i, NofWKS)
Next i
For i = 1 To NofWKS
   For j = 1 To NofTunit
      Tabpic.CurrentY = (NofWKS - i + 0.75) - 0.1
      Tabpic.CurrentX = j - 0.8
      Tabpic.Print BstRTplan(i, j)
   Next j
Next i
Tabpic.FontBold = True
For i = 1 To NofWKS
  Tabpic.CurrentX = -0.3
  Tabpic.CurrentY = (NofWKS - i + 0.75) - 0.1
  Tabpic.Print " " & i
Next i
Tabpic.CurrentY = NofWKS + 0.9
Tabpic.CurrentX = (NofTunit / 2) - 3
Tabpic.Print "Worker alocation over the production horizon:"
Tabpic.CurrentY = NofWKS
For i = 1 To NofTunit
  Tabpic.CurrentY = NofTunit + 0.4
  Tabpic.CurrentX = i - 0.8
  Tabpic.Print "t" & i
Next i
Tabpic.FontBold = False
Exit Sub
ShErr:
  MsgBox "Error!" & Err.Description
End Sub
```

```
Private Sub Schdcmd_Click()
Schdcmd.Enabled = False
caLabel.Caption = "Calculating.."
caLabel.Refresh
NofTabu = Form2.tabuLength.Text
ReDim taboMatrix(1 To NofWKS, 1 To NofTunit) As Integer
tabuIdx = 1
tabuFilled = False
calculateSkill_Motivation
BestRtplan
Dim mxMov As Integer
mxMov = MaxMov.Text
replicationNo = ReplctionNo.Text
SimulationSearch mxMov
drwTblSchdl
ComTicmd.Caption = TDfromSPT_Min
Moveno.Caption = Moveno1
caLabel.Caption = ""
caLabel.Refresh
Schdcmd.Enabled = True
End Sub

Private Sub Write_Click()
WritData3 OutptFile3.Text
WritData4 OutptFile4.Text
Writedata5 DFname6.Text
End Sub
```