國立交通大學
工業工程與管理學系

博士論文

粒子群演算法於離散最佳化問題之研究

A Study on Particle Swarm Optimization for Discrete Optimization Problems

研究生　　：徐誠佑
指導教授　：沙永傑　博士
　　　　　　陳文智　博士

中華民國九十六年五月

粒子群演算法於離散最佳化問題之研究

A Study on Particle Swarm Optimization for Discrete Optimization Problems

研 究 生： 徐誠佑 Student： Cheng-Yu Hsu

指導教授： 沙永傑 Advisor： Dr. David Yung-Jye Sha

陳文智 Dr. Wen-Chih Chen

國 立 交 通 大 學

工 業 工 程 與 管 理 學 系

博 士 論 文

A Dissertation

Submitted to Department of Industrial Engineering and Management

College of Management

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in

Industrial Engineering and Managemen

May 2007

Hsinchu, Taiwan, Republic of China

中華民國九十六年五月

粒子群演算法於離散最佳化問題之研究

學生：徐誠佑 指導教授： 沙永傑博士
陳文智博士

國立交通大學工業工程與管理系

中 文 摘 要

粒子群演算法(Particle Swarm Optimization, PSO)是一種群體搜尋最佳化演算法，於 1995 年被提出。原始的 PSO 是應用於求解連續最佳化問題。當 PSO 用來求解離散最佳化問題時，我們必須修改粒子位置、粒子移動以及粒子速度的表達方式，讓 PSO 更適於求解離散最佳化問題問題。本研究之主要貢獻為提出數種適合求解離散最佳化問題之 PSO 設計。這些新的設計和原始的設計不同且更適合求解離散最佳化問題。

在本篇論文中，我們將分別提出適合求解零壹多限制式背包問題(Multidimensional 0-1 Knapsack Problem, MKP)、零工式排程問題(Job Shop Scheduling Problem JSSP)以及開放式排程問題(Open Shop Scheduling Problem, OSSP)的 PSO。在求解 MKP 的 PSO 中，我們以零壹變數表達粒子位置，以區塊建立(building blocks)的概念表達粒子移動方式。在求解 JSSP 的 PSO 中，我們以偏好列表(preference-list)表達粒子位置，以交換運算子(swap operator)表達粒子移動方式。在求解 OSSP 的 PSO 中，我們以優先權重(priority)表達粒子位置，以插入運算子(insert operator)表達粒子移動方式。除此之外，我們在求解 MKP 的 PSO 中加入區域搜尋法(local search)，在求解 JSSP 的 PSO 與塔布搜尋(tabu search)混合，以及將求解 OSSP 的 PSO 與集束搜尋法(beam search)混合。計算結果顯示，我們的 PSO 比其它傳統的啟發式解法要來的好。

關鍵字：粒子群演算法、零壹多限制式背包問題、零工式排程問題、開放式排程問題、啟發式解法

A Study on Particle Swarm Optimization for Discrete Optimization Problems

Student：Cheng-Yu Hsu Advisor： Dr. David Yung-Jye Sha
Dr. Wen-Chih Chen

Department of Industrial Engineering and Management
National Chiao Tung University

ABSTRACT

Particle Swarm Optimization (PSO) is a population-based optimization algorithm, which was developed in 1995. The original PSO is used to solve continuous optimization problems. Due to solution spaces of discrete optimization problems are discrete, we have to modify the particle position representation, particle movement, and particle velocity to better suit PSO for discrete optimization problems. The contribution of this research is that we proposed several PSO designs for discrete optimization problems. The new PSO designs are better suit for discrete optimization problems, and differ from the original PSO.

In this thesis, we propose three PSOs for three discrete optimization problems respectively: the multidimensional 0-1 knapsack problem (MKP), the job shop scheduling problem (JSSP) and the open shop scheduling problem (OSSP). In the PSO for MKP, the particle position is represented by binary variables, and the particle movement are based on the concept of building blocks. In the PSO for JSSP, we modified the particle position representation using preference-lists and the particle movement using a swap operator. In the PSO for OSSP, we modified the particle position representation using priorities and the particle movement using an insert operator. Furthermore, we hybridized the PSO for MKP with a local search procedure, the PSO for JSSP with tabu search (TS), and the PSO for OSSP with beam search (BS). The computational results show that our PSOs are better than other traditional metaheuristics.

Keywords: Particle swarm optimization, Multidimensional 0-1 Knapsack Problem, Job shop scheduling problem, Open shop scheduling problem, Metaheuristic

誌 謝

經過四年的時間終於取得了博士學位。首先要感謝我的指導教授沙永傑老師的教導，以及共同指導教授陳文智老師協助我處理系上相關規定，讓我能夠順利畢業。也感謝其它四位口試委員在論文口試中的悉心指導，交通大學鍾淑馨教授、清華大學許棟樑教授、中華大學謝玲芬教授、雲林科技大學駱景堯教授。有了他們的悉心指導，讓這篇論文更臻完善。

感謝陪我渡過這幾年的研究室學長姊、學弟妹以及同學們，讓我留下美好的回憶。感謝陪我吃到飽的朋友們，讓我在四年內胖了十五公斤。感謝俊仁學長和嘉若學姊，這幾年來他們提供我兼課的機會，讓我的生活費有所著落。感謝在明新科技大學的學生們，讓我體驗愉快的教學經驗。感謝實驗室的戰友們，讓我在低潮的時候有發洩情緒的地方。感謝佳樺，這幾年來我一直沒辦法給她承諾，但她總是一路陪著我。

最後要感謝家人的支持，已經30多歲的我，這四年來沒有賺過一毛錢回家，讓我心中常覺虧欠。但他們還是支持我，讓我在最好的環境下完成學業。

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Research Motivations

In an optimization problem, limited resources need to be allocated for maximum profit. When an optimization problem has discrete solution space, solving such a problem amounts to making discrete choice such that an optimal solution is found among a finite or a countable infinite number of alternatives. Such problems are called discrete optimization problems. Typically, the task is complex, limiting the practical utility of combinatorial, mathematical programming and other analytical methods in solving discrete optimization problems effectively.

To find exact solutions of discrete optimization problems a branch-and-bound or dynamic programming algorithm is often used. However, many discrete optimization problems are NP-hard, which means that the problem cannot be exactly solved in a reasonable computation time. Using problem-specific information sometimes reduces search space, even though the problem is still difficult to solve exactly. Therefore, heuristic algorithms are developed to obtain the approximate optimal solution. Metaheuristic is one of the most popular and the most efficient method to obtain the approximate optimal solution. Among the meta-heuristics, particle swarm optimization (PSO) is new and extensively implemented in recent years. However, the original intent of PSO is to solve continuous optimization problems, and PSO methods that work well for discrete optimization problems are still scarce.

1.2 Research Objectives

The objective of this work is to development PSOs for three discrete optimization problems: the multidimensional 0-1 knapsack problem (MKP), the job shop scheduling problem (JSSP) and the open shop scheduling problem (OSSP).

Since the original intent of PSO is to solve continuous optimization problems, we have to modify the original PSO when we implement PSO to a discrete optimization problem. PSO can be separated several parts to discuss: position representation, particle velocity, and particle movement. We will develop various PSO designs in this work. On the other hand, the PSO developed in this work can be an example of PSO design for other discrete optimization problems.

1.3 Organization

The organization of the remaining chapters for this research is as follows. Chapter 2 reviews the literatures of the background of the multidimensional 0-1 knapsack problem, shop scheduling problems and PSO. Chapter 3 infers the possibly success factors of PSO design. Chapter 4 shows a PSO for MKP, chapter 5 shows a PSO for JSSP, and chapter 6 shows a PSO for OSSP. In chapter 7 we draw our conclusion and indicate the direction for further research.

CHAPTER 2

LITERATURE REVIEW

2.1 Particle Swarm Optimization

Particle swarm optimization (PSO) was developed by Kennedy and Eberhart (1995). The original intention was to simulate the movement of organisms in a bird flock or fish school, and it has since been introduced as an optimization technique. PSO is a population-based optimization algorithm. Each particle is an individual, and the swarm is composed of particles. The relationship between swarm and particles in PSO is similar to the relationship between population and chromosomes in genetic algorithm (GA).

In PSO, the problem solution space is formulated as a search space. Each position in the search space is a correlated solution of the problem. For example, when PSO is applied to a continuous optimization problem with d variables, the solution space can be formulated as a d dimensional search space, and the value of j^{th} variable is formulated as the position on j^{th} dimension. Particles cooperate to find the best position (best solution) in the search space (solution space). The particle movement is mainly affected by three factors: inertia, particle best position (*pbest*), and global best position (*gbest*). The inertia is the velocity of the particle in the latest iteration, and it can be controlled by inertia weight. The intention of the inertia is to prevent particles from moving back to their current positions. The *pbest* position is the best solution found by each particle itself so far, and each particle has its own *pbest* position. The *gbest* position is the best solution found by the whole swarm so far.

Each particle moves according to its velocity. The velocity is randomly generated toward *pbest* and *gbest* positions. For each particle k and dimension j, the velocity and position of particles can be updated by the following equations:

$$v_{kj} \leftarrow w \times v_{kj} + c_1 \times rand_1 \times (pbest_{kj} - x_{kj}) + c_2 \times rand_2 \times (gbest_j - x_{kj}) \qquad (2.1)$$

$$x_{kj} \leftarrow x_{kj} + v_{kj} \qquad (2.2)$$

In equation (2.1) and equation (2.2), v_{kj} is the velocity of particle *k* on dimension *j*, which value is limited to the parameter $V_{\max}$, that is, $|v_{kj}| \le V_{\max}$. The x_{kj} is the position of particle *k* on dimension *j*, which value is limited to the parameter $X_{\max}$, that is, $|x_{kj}| \le X_{\max}$. The $pbest_{kj}$ is the *pbest* position of particle *k* on dimension *j*, and $gbest_j$ is the *gbest* position of the swarm on dimension *j*. The inertia weight w was first proposed by Shi and Eberhart (1998a, 1998b), and it is used to control exploration and exploitation. The particles maintain high velocities with a larger w, and low velocities with a smaller w. A larger w can prevent particles from becoming trapped in local optima, and a smaller w encourages particles exploiting the same search space area. The constants c_1 and c_2 are used to decide whether particles prefer moving toward a *pbest* position or *gbest* position. The $rand_1$ and $rand_2$ are random variables between 0 and 1. The process of PSO is shown as Figure 2.1.

Initialize a population of particles with random positions and velocities on *d* dimensions in the search space.

repeat

for each particle *k* ***do***

Update the velocity of particle *k*, according to equation (1).

Update the position of particle *k*, according to equation (2).

Map the position of particle *k* in the solution space and evaluate its fitness value according to the desired optimization fitness function.

Update *pbest* and *gbest* position if necessary.

end for

until a criterion is not met, usually a sufficient good fitness or a maximum number of iterations.

Figure 2.1 The process of particle swarm optimization.

The original PSO is suited to a continuous solution space. Therefore, the applications of PSO for discrete optimization problems are still scarce. Table 2.1 shows the references of PSO for discrete optimization problems.

Table 2.1 References of PSO for discrete optimization problems

Problem	References
Binary Unconstrained Optimization	Kennedy & Eberhart (1997) A first discrete version of PSO for binary variables. Rastegar et al. (2004) Based on Kennedy & Eberhart (1997), hybridized with learning automata.
Constrained layout optimization	Li (2004) Represent a layout problem by continuous variables.
Multi-objective task allocation	Yin et al. (2007a) Particle represented by integer variables, which indicates the index of the allocated processor for each module.
Task assignment problem	Salman et al. (2002) The positions of tasks are represented by continuous variables. Yin et al. (2007b) Hybridized with a parameter-wise hill-climbing heuristic.
Traveling salesman problem	Wang et al. (2003) Applied sequential ordering representation and swap operator. Pang et al. (2004) Represent the particle position by a fuzzy matrix. Zhi et al. (2004) The particles movement is based on a one-point crossover.
Vehicle routing problem	Wu et al. (2004) Applied swap operator and 2-opt local search.

Table 2.1 (cont.)

Problem	References
Scheduling Problems:	
– Assembly scheduling	Allahverdi & Al-Anzi (2006) Hybridized with tabu search.
– Flow shop scheduling	Lian et al. (2006) Proposed three crossover operators. Tasgetiren et al. (2007) Real number representation and local search. Liao et al. (2007) Represent the operation sequence by 0-1 variables.
– Job shop scheduling	Lian et al. (2006) Proposed four crossover operators.
– Multi-objective flexible job-shop scheduling	Xia & Wu (2005) Hybridized with simulated annealing.
– Resource constraint project scheduling	Zhang & Li (2006) Zhang & Li (2007) Compared priority based representation and sequential ordering representation.

2.2 Multidimensional 0-1 Knapsack Problem

The multidimensional 0-1 knapsack problem (MKP) is a well-known NP-hard problem. The problem can be formulated as:

$$
\begin{array}{lll}
\text{maximize} & \sum_{j=1}^{n} p_j x_j \,, & \\
\text{subject to} & \sum_{j=1}^{n} r_{ij} x_j \le b_i \,, & \textit{for} \;\; i = 1, \ldots, m \,, \\
 & x_j \in \{0,1\} \,, & \textit{for} \;\; j = 1, \ldots, n \,, \\
\text{with} & p_j > 0 \,, & \textit{for all } j, \\
 & 0 \le r_{ij} \le b_i \,, & \textit{for all } i, j, \\
 & b_i < \sum_{j=1}^{n} r_{ij} \,, & \textit{for all } i.
\end{array}
$$

Where m is the number of knapsack constraints and n is the number of items. Each item j requires r_{ij} units of resource consumption in the i^{th} knapsack and yields p_j units of profit upon inclusion. The goal is to find a subset of items that yields maximum profit without exceeding resource capacities. The MKP can be seen as a general model for any kind of binary problems with positive coefficients, and it can be applied to many problems such as cargo loading, capital budgeting, project selection, etc. The most recent surveys on MKP can be found in (Fréville, 2004) and (Fréville and Hanafi, 2005).

The MKP is an NP-hard problem, so it cannot be exactly solved in a reasonable computation time for large instances. However, metaheuristics can obtain approximate optimal solutions in a reasonable computation time. For that reason, metaheuristics for MKP such as simulated annealing (SA) (Drexl, 1988), tabu search (TS) (Glover and Kochenberger, 1996; Hanafi and Fréville; 1998; Vasquez and Hao, 2001; Vasquez and Vimont, 2005), and genetic algorithm (GA) (Chu and Beasley, 1998) have arisen during the last decade.

2.3 Job Shop Scheduling Problem

The job shop scheduling problem (JSSP) is one of the most difficult combinatorial optimization problems. The JSSP can be briefly stated as follows (French, 1982; Gen & Cheng, 1997). There are n jobs to be processed through m machines. We shall suppose that each job must pass through each machine once and once only. Each job should be processed through the machines in a particular order, and there are no precedence constraints among different job operations. Each machine can process only one job at a time, and it cannot be interrupted. Furthermore, the processing time is fixed and known. In this work, the problem is to find a schedule to minimize the makespan ($C_{\max}$), that is, the time required to complete all jobs. The constraints in the classical JSSP is listed as follows (Bagchi, 1999):

- No two operations of one job occur simultaneously.
- No pre-emption (i.e. process interruption) of an operation is allowed.
- No job is processed twice on the same machine.
- Each job is processed to its completion, though there may be waits and delays between the operations performed.
- Jobs may be started at any time; hence no release time exists.
- Jobs must wait for the next machine to be available.
- No machine may perform more than one operation at a time.
- Set-up times for the operations are sequence-independent and included in processing times.
- There is only one of each type of machine.
- Machines may be idle within the schedulable period.
- Machines are available at any time.

- The technological (usually related to processing) constraints are known in advance and are immutable.

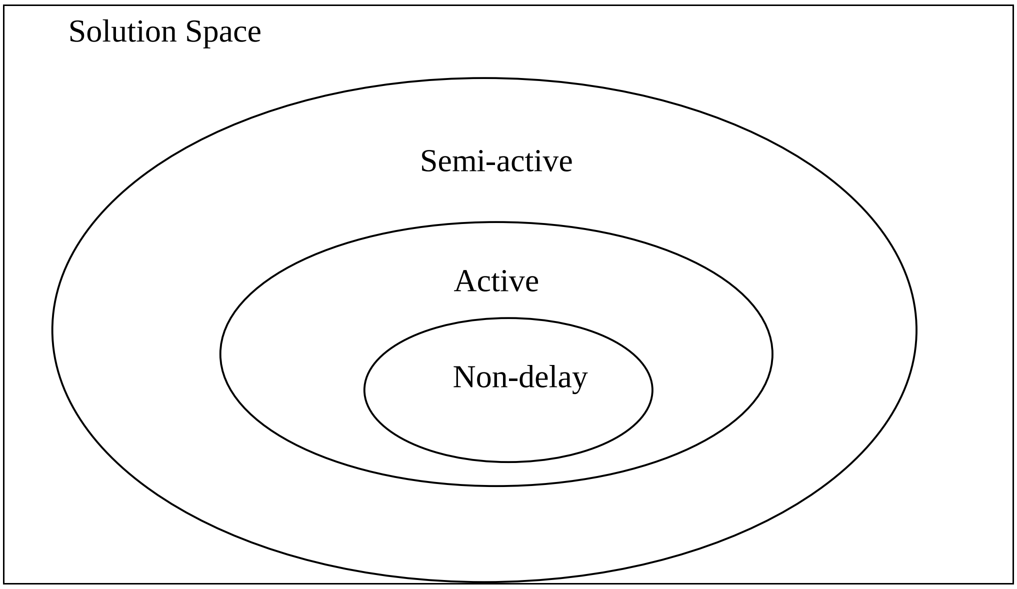


Figure 2.2 The relationship of semi-active, active, and nondelay schedules.

In JSSP, there are three distinct schedules can be identified as follows:

- ***Semi-active schedule***: in an active schedule, the processing sequence is such that no operation can be started any earlier without changing the operation sequence on a machine.
- ***Active schedule***: in an active schedule, the processing sequence is such that no operation can be started any earlier without delaying some other operation.
- ***Nondelay schedule***: in a nondelay schedule, no machine is kept idle at a time when it could begin processing other operations.

Figure 2.2 shows the relationship of semi-active, active, and nondelay schedules. The optimal JSSP solution should be an active schedule. To reduce the search solution space, the tabu search proposed by Sun et al. (1995) searches solutions within the set of active schedules. Gonçalves and Beirão (1999) proposed the concept of parameterized active schedules. The main purpose of parameterized active schedules is to reduce the search area but not to exclude the optimal solution. The basic idea of parameterized active schedules is to control the search area by controlling the delay times that each operation is allowed. If all of the delay times are equal to zero, the set of parameterized active schedules is equivalent to non-delay schedules. On the contrary, if all of the delay times are equal to infinity, the set of parameterized active schedules is equivalent to the active schedules.

Garey et al. (1976) demonstrated that JSSP is NP-hard, so it cannot be exactly solved in a reasonable computation time. Many meta-heuristics have been developed in the last decade to solve JSSP, such as *simulated annealing* (SA) (Lourenço, 1995), *tabu search* (TS) (Sun et al., 1995; Nowicki & Smutnicki, 1996; Pezzella & Merelli, 2000), and *genetic algorithm* (GA) (Bean, 1994; Kobayashi et al., 1995; Wang & Zheng, 2001; Gonçalves et al., 2005).

2.4 Open Shop Scheduling Problem

The open shop scheduling problem (OSSP) can be stated as follows (Gonzalez & Sahni, 1976): there is a set of n jobs that have to be processed on a set of m machines. Every job consists of m operations, each of which must be processed on a different machine for a given process time. The operations of each job can be processed in any order. At any time, at most one operation can be processed on each machine, and at most one operation of each job can be processed. In this research, the problem is to find a non pre-emptive schedule to minimize the makespan ($C_{\max}$), that is, the time required to complete all jobs.

The constraints in the classical OSSP are similar to the classical JSSP but there are no precedence constraints among the same job operations. The OSSP is NP-hard for $m \geq 3$ (Gonzalez & Sahni, 1976), so it cannot be exactly solved in a reasonable computation time. Guéret and Prins (1998) proposed two fast heuristics, the results of which are better than other classical heuristics. Domdorf et al. (2001) proposed a branch-and-bound method, which is the current best method to solve OSSP exactly. Many metaheuristic algorithms have been developed in the last decade to solve OSSP, such as *simulated annealing* (SA) (Liaw, 1999), *tabu search* (TS) (Alcaide & Sicilia, 1997; Liaw, 1999), *genetic algorithm* (GA) (Liaw, 2000; Prins, 2000), *ant colony optimization* (ACO) (Blum, 2005), and *neural network* (NN) (Colak & Agarwal, 2005).

CHAPTER 3

DEVELOPING A PARTICLE SWARM OPTIMIZATION FOR A DISCRETE OPTIMIZATION PROBLEM

The original PSO is suited to a continuous solution space. We have to modify the original PSO in order to better suit it to discrete optimization problems. In this chapter, we will discuss the probably success factors to develop a PSO design for a discrete optimization problem. We separated a PSO design into several parts to discuss: particle position representation, particle velocity, particle movement, decoding operator, and other search strategies.

3.1 Particle Position Representation

PSO represents the solutions by particle positions. There are various particle position representations for a discrete optimization problem. How to represent solutions by particle positions is a research topic when we develop a PSO design.

Generally, the Lamarckian property is used to discriminate between good and bad representations. The Lamarckian property is that the offspring can inherit goodness from its parents. For example, if there are six operations to be sorted on a machine, and we implement the random key representation (Bean, 1994) to represent a sequence, there are two positions of two particle positions as follows:

position 1: [0.25, 0.27, 0.21, 0.24, 0.26, 0.23]

position 2: [0.22, 0.25, 0.23, 0.26, 0.24, 0.21]

Then the operation sequence can be generated by sort the operations according to the increasing order of their position values as follows:

permutation 1: [3 6 4 1 5 2]

permutation 2: [6 1 3 5 2 4]

We can find that these two permutations are quite different even though their positions are very close to each other. This is because the location in the permutation of one operation depends on the position values of other operations. Hence, the random key representation has no Lamarckian.

If we directly implement the original PSO design (i.e. the particles search solutions in a continuous solution space) to a scheduling problem, we can implement the random key representation to represent a sequence of operations on a machine. However, the PSO will be more efficient if the particle position representation is with higher Lamarckian.

3.2 Particle Velocity and Particle Movement

The particle velocity and particle movement are designed for the specific particle position representation. In each iteration, a particle moves toward *pbest* and *gbest* positions, that is, the next particle position is determined by current position, *pbest* position, and *gbest* position. Furthermore, particle moves according to its velocity and movement mechanisms. Each particle moves from current position (solution) to one of the neighborhood positions (solutions). Therefore, the particle movement mechanism should be designed according the neighborhood structure. The advantage of neighborhood designs can be estimated by following properties (Mattfeld, 1996):

- ***Correlation***: the solution resulting from a move should not differ much from the starting one.
- ***Feasibility***: the feasibility of a solution should be preserved by all moves.

- ***Improvement***: all moves in the neighborhood should have a good chance to improve the objective of a solution.
- ***Size***: the number of moves in the neighborhood should be reasonably small to avoid excessive computational cost of their evaluation.
- ***Connectivity***: it should be possible to reach the optimal solution from any starting one by performing a finite number of neighborhood moves.

We believe that the PSO will be more efficient if we design the particle velocity and the particle movement mechanisms according to these properties.

3.3 Decoding Operator

Decoding operator is used to decode a particle position into a solution. The decoding operator is designed according the specific particle position representation and the characteristics of the problem. A superior decoding operator can map the positions to the solution space in a smaller region but not excluding the optimal solution. In chapter 6, we designed four decoding operators for OSSP. The results show that the decoding operator design extremely influences the solution quality.

3.4 Other Search Strategies

. ***Diversification Strategy***

We can also consider implementing other search strategies. The purpose of most search strategies is to control the intensification and the diversification. One of the search strategies is the structure of *gbest* and *pbest* solutions. In the original PSO design, each particle has its own *pbest* solution and the swam has only one *gbest* solution. Eberhart and Shi (2001) show a “local” version of the particle swarm. In this version, particles have information only of their own and their neighbors’ bests, rather that that of the entire group. Instead of moving toward a kind of stochastic average of *pbest* and *gbest* (the best location of the entire group), particles move toward points defined by *pbest* and "*lbest*," which is the index of the particle with the best evaluation in the particle's neighborhood.

In this research, we proposed a *diversification strategy*. In this strategy, the *pbest* solution of each particle is not the best solution found by the particle itself, but one of the best N solutions found by the swarm so far where N is the size of the swarm.

. ***Selection Strategy***

Angeline (1998) proposed a selection strategy, which is performed as follows. After all the particles move to new positions, select the s best particles. The better particle set $S = \{k_1, k_2, \ldots, k_s\}$ replaces the positions and velocities of the other particles. The addition of this selection strategy should provide the swarm with a more exploitative search mechanism that should find better optima more consistently.

In chapter 4, we modified the method proposed by Angeline (1998) based on the concept of building blocks where a block is part of a solution, and a good solution should include some superior blocks. The concept of building blocks is that if we can precisely find out the superior blocks and accumulate the superior blocks in the population, the genetic algorithm will perform better (Goldberg, 2002).

3.5 The Process to Develop a New Particle Swarm Optimization

As mentioned above, we separated PSO into several parts. The particle position representation determines the program data structure and other parts are designed for the specific particle position representation. Therefore, the first step of developing a new PSO is to determine the particle position representation. Then design particle velocity, particle movement and decoding operator for the specific particle position representation. Finally implement some search strategies for further improving solution quality. Figure 3.1 shows the process to develop a new particle swarm optimization. All the PSOs in this research are developed by the process described as Figure 3.1.

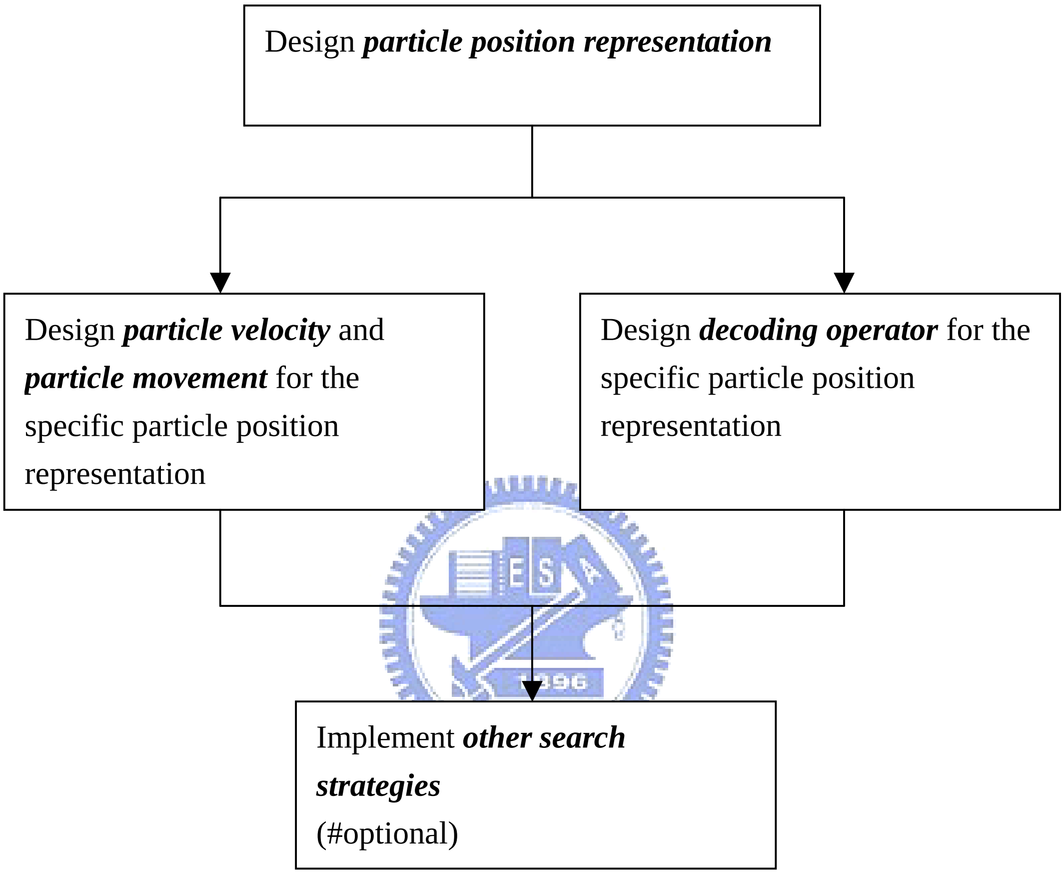


Figure 3.1 The process to develop a new particle swarm optimization

CHAPTER 4

A DISCRETE BINARY PARTICLE SWARM OPTIMIZATION FOR THE MULTIDIMENSIONAL 0-1 KNAPSACK PROBLEM

Kennedy and Eberhart (1997) proposed a discrete binary particle swarm optimization (DPSO), which was designed for discrete binary solution space. Rastegar et al. (2004) proposed another DPSO based on learning automata. In both of the DPSOs, when particle k moves, its position on the j^{th} variable x_{kj} equals 0 or 1 at random. The probability that x_{kj} equals 1 is obtained by applying a sigmoid transformation to the velocity v_{kj} $(1/1+\exp(-v_{kj}))$. In the above two DPSOs, the positions and velocities of particles can be updated by the following equations:

$$v_{kj} \leftarrow v_{kj} + c_1 \times rand_1 \times (pbest_{kj} - x_{kj}) + c_2 \times rand_2 \times (gbest_j - x_{kj}) \tag{4.1}$$

$$if\ (rand < (1/1+\exp(-v_{kj}))\ then\ x_{kj} \leftarrow 1\ \ else\ x_{kj} \leftarrow 0 \tag{4.2}$$

where rand, $rand_1$, and $rand_2$ are random numbers between 0 and 1. The value of v_{kj} is limited by a value $V_{\max}$, a parameter of the algorithm, that is, $|v_{kj}| \le V_{\max}$. For instance, if $V_{\max}$ = 6, the probability that x_{kj} equals 1 will be limited between 0.9975 and 0.0025.

The DPSOs proposed by Kennedy and Eberhart (1997) and Rastegar et al. (2004) are designed for discrete binary optimization problems with no constraint. However, there are resource constraints in MKP. If we want to solve MKP by DPSO, we have to modify DPSO to fit the MKP characteristics.

There are two main differences between the DPSO in this chapter and the DPSOs in previous research: (i) particle velocity and (ii) particle movement. The particle velocity is modified based on the tabu list and the concept of building blocks, and then the particle movement is modified based on the crossover and mutation of the genetic algorithm (GA). Besides, we applied the repair operator to repair solution infeasibility, the diversification strategy to prevent particles becoming trapped in local optima, the selection strategy to exchange good blocks between particles, and the local search to further improve solution quality. The computational results show that our DPSO effectively solves MKP and better than other traditional algorithms.

4.1 Particle Position Representation

In out DPSO, the particle position is represented by binary variables. For a MKP with n items, we represent the particle k position by n binary variables, i.e.

$$x_k = [x_{k1}, x_{k2}, \ldots, x_{kn}]$$

Where $x_{kj} \in \{0, 1\}$ denotes the value of j^{th} variable of particle k's solution. Each time we start a run, DPSO initializes a population of particles with random positions, and initializes the *gbest* solution by a surrogate duality approach (Pirkul, 1987). We determine the pseudo-utility ratio $u_j = p_j / \sum_{i=1}^{m} y_i r_{ij}$ for each variable, where y_i is the shadow price of the i^{th} constraint in the LP relaxation of the MKP. To initialize the *gbest* solution, we set $gbest_j \leftarrow 0$ for all variable j and then add variables ($gbest_j \leftarrow 1$) into the *gbest* solution by descending order of u_j as much as possible without violating any constraint.

Initializing the *gbest* solution has two purposes. The first is to improve the consistency of run results. Because the solutions of DPSO are generated randomly, the computational results will be different in each run. If we give DPSO the same initial point of *gbest* solution in each run, it may improve result consistency. The second purpose is to improve result quality. Similar to other local search approaches, a good initial solution can accelerate solution convergence with better results.

4.2 Particle Velocity

When PSO is applied to solve problems in a continuous solution space, due to inertia, the velocity calculated by equation (4.1) not only moves the particle to a better position, but also prevents the particle from moving back to the previous position. The larger the inertia weight, the harder the particle backs to the current position. The DPSO velocities proposed by Kennedy and Eberhart (1997) and Rastegar et al. (2004) move particles toward the better position, but cannot prevent the particles from being trapped in local optima.

We modified the particle velocity based on the tabu list, which is applied to prevent the solution from being trapped in local optima. In our DPSO, each particle has its own tabu list, the velocity. There are two velocity values, v_{kj} and v'_{kj}, for each variable x_{kj}. If x_{kj} changes when particle *k* moves, we set $v_{kj} \leftarrow 1$ and $v'_{kj} \leftarrow x_{kj}$. When v_{kj} equals 1, it means that x_{kj} has changed, variable *j* was added into the tabu list of particle *k*, and we should not change the value of x_{kj} in the next few iterations. Therefore, the velocity can prevent particles from moving back to the last position in the next few iterations. The value of v'_{kj} is used to record the value of x_{kj} after the value of x_{kj} has been changed. The set of variable v'_{kj} is a "block" which is a part of a solution that particle *k* obtained from the *pbest* solution and *gbest* solution. It is applied to the selection strategy with the concept of building blocks that we will describe in section 4.6.

In our DPSO, we also implement inertia weight w to control particle velocities where w is between 0 and 1. We randomly update velocities at the beginning of each iteration. For each particle k and j^{th} variable, if v_{kj} equals 1, v_{kj} will be set to 0 with probability (1-w). This means that if variable x_{kj} is in the tabu list of particle k, variable x_{kj} will be dropped from the tabu list with probability (1-w). Moreover, the exploration and exploitation can be controlled by w. The variable x_{kj} will be held in the tabu list for more iterations with a larger w and vice versa. The pseudo code of updating velocities is given in Figure 4.1.

```
for each particle k and variable j do
    rand ~ U(0,1)
    if (v_kj = 1) and (rand ≥ w) then
        v_kj ← 0
    end if
end for
```

Figure 4.1 Pseudo code of updating velocities.

4.3 Particle Movement

In the DPSO we proposed, particle movement is similar to the crossover and mutation of GA. When particle k moves, if x_{kj} is not in the tabu list of particle k (i.e. v_{kj} =0), the value of x_{kj} will be set to $pbest_{kj}$ with probability c_1 (if $x_{kj} \neq pbest_{kj}$), set to $gbest_j$ with probability c_2 (if $x_{kj} \neq gbest_j$), set to (1-x_{kj}) with probability c_3, or not changed with probability (1-c_1-c_2-c_3). Where c_1, c_2, and c_3 are parameters of the algorithm with $c_1 + c_2 + c_3 \leq 1$ and $c_i \geq 0$, i=1, 2, 3.

Since the value of x_{kj} may be changed by repair operator or local search procedure (we will describe them in section 4.4 and in section 4.7 respectively), if x_{kj} is in the tabu list of particle k (i.e. v_{kj} = 1), the value of x_{kj} will be set to the value of v'_{kj} which is the latest value that x_{kj} obtained from the $pbest$ solution or $gbest$ solution. At the same time, if the value of x_{kj} changes, we update v_{kj} and v'_{kj} as we mentioned in section 4.2. The pseudo code of particle movement is given in Figure

4.2.

for $j \leftarrow 1$ **to** n **do**

$\quad rand \sim U(0,1)$

$\quad$ **if** $(v_{kj} = 0)$ **then**

$\quad\quad$ **if** $(rand \le c_1)$ **and** $(x_{kj} \ne pbest_{kj})$ **then**

$\quad\quad\quad x_{kj} \leftarrow pbest_{kj};\ v_{kj} \leftarrow 1;\ v'_{kj} \leftarrow x_{kj}$

$\quad\quad$ **if** $(c_1 < rand \le c_1 + c_2)$ **and** $(x_{kj} \ne gbest_j)$ **then**

$\quad\quad\quad x_{kj} \leftarrow gbest_j;\ v_{kj} \leftarrow 1;\ v'_{kj} \leftarrow x_{kj}$

$\quad\quad$ **if** $(c_1 + c_2 < rand \le c_1 + c_2 + c_3)$ **then**

$\quad\quad\quad x_{kj} \leftarrow (1 - x_{kj});\ v_{kj} \leftarrow 1;\ v'_{kj} \leftarrow x_{kj}$

$\quad$ **else**

$\quad\quad x_{kj} \leftarrow v'_{kj}$

$\quad$ **end if**

end for

Figure 4.2 Pseudo code of particle movement.

4.4 Repair Operator

After a particle generates a new solution, we apply the repair operator to repair solution infeasibility and to improve it. There are two phases to the repair operator. The first is the drop phase. If the particle generates an infeasible solution, we need to drop ($x_{kj} \leftarrow 0$, *if* $x_{kj} = 1$) some variables to make it feasible. The second phase is the add phase. If the particle finds a feasible solution, we add ($x_{kj} \leftarrow 1$, *if* $x_{kj} = 0$) more variables to improve it. Each phase is performed twice: the first time we consider the particle velocities, and the second time we do not consider the particle velocities.

Similar to initializing the *gbest* solution described in section 4.1, we applied the Pirkul (1987) surrogate duality approach to determine the variable priority for adding or dropping. First, we determine the pseudo-utility ratio $u_j = p_j / \sum_{i=1}^{m} y_i r_{ij}$ for each variable, where y_i is the shadow price of the i^{th} constraint in the LP relaxation of the MKP. We drop variables by ascending order of u_j until the solution is feasible, and then we add variables by descending order of u_j as much as possible without violating any constraint. The pseudo code of repair operator is given in Figure 4.3.

R_i = *the accumulated resources of constraint i*

$U \leftarrow$ *permutation of* (1,2,…,n) *with* $u_{U[j]} \geq u_{U[j+1]}$ (j = 1,…, n-1)

$R_i \leftarrow \sum_{j=1}^{n} r_{ij} x_{kj}, \forall i;$

// Drop phase begin

for $j \leftarrow n$ **to** 1 **do**

    if $(x_{kU[j]} = 1)$ ***and*** $(R_i > b_i,$ *for any i*$)$ ***and*** $(v_{kU[j]} = 0)$ ***then***

        $x_{kU[j]} \leftarrow 0;$

        $R_i \leftarrow (R_i - r_{iU[j]}), \forall i;$

    end if

end for

for $j \leftarrow n$ **to** 1 **do**

    if $(x_{kU[j]} = 1)$ ***and*** $(R_i > b_i,$ *for any i*$)$ ***and*** $(v_{kU[j]} = 1)$ ***then***

        $x_{kU[j]} \leftarrow 0;$

        $R_i \leftarrow (R_i - r_{iU[j]}), \forall i;$

    end if

end for

// Drop phase end

// Add phase begin

for $j \leftarrow 1$ **to** n **do**

    if $(x_{kU[j]} = 0)$ ***and*** $(R_i + r_{iU[j]} \leq b_i, \forall i)$ ***and*** $(v_{kU[j]} = 0)$ ***then***

        $x_{kU[j]} \leftarrow 1;$

        $R_i \leftarrow (R_i + r_{iU[j]}), \forall i;$

    end if

end for

for $j \leftarrow 1$ **to** n **do**

    if $(x_{kU[j]} = 0)$ ***and*** $(R_i + r_{iU[j]} \leq b_i, \forall i)$ ***and*** $(v_{kU[j]} = 1)$ ***then***

        $x_{kU[j]} \leftarrow 1;$

        $R_i \leftarrow (R_i + r_{iU[j]}), \forall i;$

    end if

end for

// Add phase end

Figure 4.3 Pseudo code of repair operator

4.5 The Diversification Strategy

If the *pbest* solutions are all the same, the particles will be trapped in local optima. To prevent such a situation, we propose a diversification strategy to keep the *pbest* solutions different. In the diversification strategy, the *pbest* solution of each particle is not the best solution found by the particle itself, but one of the best N solutions found by the swarm so far where N is the size of the swarm.

The diversification strategy is performed according to the following process. After all of the particles generate new solutions, for each particle, compare the particle's fitness value with *pbest* solutions. If the particle's fitness value is better than the worst *pbest* solution and the particle's solution is not equal to any of the *pbest* solutions, replace the worst *pbest* solution with the solution of the particle. At the same time, if the particle's fitness value is better than the fitness value of the *gbest* solution, replace the *gbest* solution with the solution of the particle. The pseudo code of updating *pbest* solutions is given in Figure 4.4.

k^* *is the index of the worst pbest solution*
for $k \leftarrow 1$ to N ***do***
    $k^* = \arg\min_{\forall k'} \{ f(pbest_{k'}) \}$;
    if $(f(x_k) > f(pbest_{k^*}))$ ***then***
        if $(x_k \neq pbest_{k'}, \forall k')$ ***then***
            $pbest_{k^*} \leftarrow x_k$;
        end if
        if $(f(x_k) > f(gbest))$ ***then***
            $gbest \leftarrow x_k$;
        end if
    end if
end for

Figure 4.4 Psudo code of updating *pbest* solutions.

4.6 The Selection Strategy

Angeline (1998) proposed a selection strategy, which is performed as follows. After all the particles move to new positions, select the s best particles. The better particle set $S = \{k_1, k_2, \ldots, k_s)$ replaces the positions and velocities of the other particles. The addition of this selection strategy should provide the swarm with a more exploitative search mechanism that should find better optima more consistently.

We modified the method proposed by Angeline (1998) based on the concept of building blocks where a block is part of a solution, and a good solution should include some superior blocks. The concept of building blocks is that if we can precisely find out the superior blocks and accumulate the superior blocks in the population, the genetic algorithm will perform better (Goldberg, 2002).

```
Find out the s best particles  S = {k_1, k_2, ..., k_s)
l ← 1;
 for k ← 1 to N do
      if (k ∉ S) then
           v_k ← v_{k_l};
           v'_k ← v'_{k_l};
           if (l > s) then
                l ← 1;
           else
                l ← l + 1;
           end if
      end if
end for
```

Figure 4.5 Pseudo code of selection strategy.

In our DPSO, the velocities $v'_k = \{v'_{k1}, v'_{k2}, \ldots, v'_{kn}\}$ is a block that particle k obtained from *pbest* solution and *gbest* solution in each iteration. The v'_k may be a superior block if the solution of particle k is better then others. Therefore, in our modified selection strategy, the better particle set S only replaces the velocities (i.e.

v_k and v'_k) of the other particles. The pseudo code of selection strategy is given in Figure 4.5.

4.7 Local Search

We implement a local search procedure after a particle generates a new solution for further improved solution quality. The classical add/drop neighborhood is that we remove a variable from the current solution and add another variable to it without violating any constraint at the same time. We modified the neighborhood with the concept of building blocks to reduce the neighborhood size. We focus on the block when we implement a local search. The variables are classified to 4 sets: $J_0 = \{j \mid x_{kj} = 0\}$, $J_1 = \{j \mid x_{kj} = 1\}$, $J'_0 = \{j \mid x_{kj} = 0 \wedge v_{kj} = 1\}$, and $J'_1 = \{j \mid x_{kj} = 1 \wedge v_{kj} = 1\}$. The modified neighborhood is defined as follows: add (or drop) one variable from J'_0 (or J'_1) and drop (or add) one variable from J_1 (or J_0) without violating any constraint at the same time. In our experiment, the size of the modified neighborhood is about twenty times smaller then the classical one. Besides, we add variables by descending order of p_j as much as possible without violating any constraint after the add/drop process.

R_i = *the accumulated resources of constraint i*

$P \leftarrow$ *permutation of* (1,2,…,n) *with* $p_{P[j]} \geq p_{P[j+1]}$ (j = 1,…, n-1)

$R_i \leftarrow \sum_{j=1}^{n} r_{ij} x_{kj}, \forall i$

for *times* $\leftarrow 1$ **to** 4 **do**

 // Add/drop local search begin

 $j_0^* \leftarrow 0;\ \ j_1^* \leftarrow 0;\ \ f^* \leftarrow f(x_k);$

 for $j' \leftarrow 1$ **to** n **do**

  if $(j' \in J_1')$ **then**

   $j_0 = \arg\max_{\forall j}\{p_j \mid j \in J_0 \text{ and } R_i - r_{ij'} + r_{ij} \leq b_i, \forall i\}$

   if $(f(x_k) - p_{j'} + p_{j_0} > f^*)$ **then**

    $j_0^* \leftarrow j';\ \ j_1^* \leftarrow j_0;\ \ f^* \leftarrow (f(x_k) - p_{j'} + p_{j_0});$

   end if

  end if

  if $(j' \in J_0')$ **then**

   $j_1 = \arg\min_{\forall j}\{p_j \mid j \in J_1 \text{ and } R_i + r_{ij'} - r_{ij} \leq b_i, \forall i\}$

   if $(f(x_k) + p_{j'} - p_{j_1} > f^*)$ **then**

    $j_0^* \leftarrow j_1;\ \ j_1^* \leftarrow j';\ \ f^* \leftarrow (f(x_k) + p_{j'} - p_{j_1});$

   end if

  end if

 end for

 $x_{k j_0^*} \leftarrow 0;\ \ x_{k j_1^*} \leftarrow 1;\ \ R_i \leftarrow (R_i + r_{i j_1^*} - r_{i j_0^*}), \forall i;$

 // Add/drop local search end

 // Add more variables begin

 for $j \leftarrow 1$ **to** n **do**

  if $(x_{kP[j]} = 0)$ **and** $(R_i + r_{iP[j]} \leq b_i, \forall i)$ **then**

   $x_{kP[j]} \leftarrow 1;\ \ R_i \leftarrow (R_i + r_{iP[j]}), \forall i;$

  end if

 end for

 // Add more variables end

end for

Figure 4.6 Pseudo code of local search procedure.

We do not repeat the local search until the solution reaches the local optima. The local search procedure is performed four times at most for reducing the computation time and preventing being trapped in local optima. The pseudo code of local search is given in Figure 4.6.

4.8 Computational Results

Our DPSO was tested on the problems proposed by Chu and Beasley (1998). These problems are available on the OR-Library web site (Beasley, 1990) (URL: http:// people.brunel.ac.uk/~mastjjb/jeb/info.html). The number of constraints m was set to 5, 10, and 30, and the number of variables n was set to 100, 250, and 500. For each m-n combination, thirty problems are generated, and the tightness ratio α ($\alpha = b_i / \sum_{j=1}^{n} r_{ij}$) was set to 0.25 for the first ten problems, to 0.5 for the next ten problems, and to 0.75 for the remaining problems. Therefore, there are 27 problem sets for different n-m-α combinations, ten problems for each problem set, and 270 problems in total.

The program was coded in Visual C^{++}, optimized by speed, and run on an AMD Athlon 1800+ PC. The numeric parameters are set to $c_1 = 0.7$, $c_2 = 0.1$, $c_3 = 1/n$, N = 100, and s = 20. The inertia weight w is decreased linearly from 0.7 to 0.5 during a run. All of the numeric parameters are determined empirically. There are two versions of our DPSO. The first version, DPSO, does not implement the local search procedure, and the second, DPSO+LS, implements the local search procedure. Each run will be terminated after 10,000 iterations on DPSO and 15,000 iterations on DPSO+LS, respectively.

The program performs 10 runs for each of the 270 problems. The term ‘Best’ is applied to the best solution obtained in 10 runs of each problem and averaged according to the 10 problems of the problem set. The term ‘Average’ is applied to the average solution of 10 runs of each problem and averaged according to the 10 problems of the problem set.

t^* is the average time in seconds that the DPSO or DPSO+LS takes to first reach the final best solution in a run, and T is the total time in seconds that the DPSO or DPSO+LS takes before termination in a run. The surrogate duality was pre-calculated by MATLAB, so the computation times in Table 4.1 do not include the computation time of calculating surrogate duality. The average computation time for solving LP relaxation problems was less then 1 CPU second, so the surrogate duality calculation is not important to computation time.

The computational results are shown in Table 4.1. The algorithms have different computation times, so we compared DPSO with GA (Chu and Beasley, 1998) since their computation times are similar. The computational results show that DPSO performed better than GA (Chu and Beasley, 1998) in 20 of 27 problem sets.

We compared DPSO+LS with Fix+Cuts (Osorio et al., 2002) and LP+TS (Vasquez and Hao, 2001). The Fix+Cuts (Osorio et al., 2002) takes 3 hours on a Pentium III 450 MHz PC for each run, and LP+TS (Vasquez and Hao, 2001) takes about 20-40 minutes on a Pentium III 500 MHz PC for each run. Our DPSO+LS takes less than 8 minutes on the largest problems and our machine is about four times faster than Fix+Cuts (Osorio et al., 2002) and LP+TS (Vasquez and Hao, 2001), so the computation time that DPSO+LS takes is similar to LP+TS (Vasquez and Hao, 2001) and much less than Fix+Cuts (Osorio et al., 2002). The computational results show that DPSO+LS performed better than Fix+Cuts (Osorio et al., 2002) in 15 of 27 problem sets, and better than LP+TS (Vasquez and Hao, 2001) in all of the 9 largest problem sets.

Table 4.1 Computational results

Problem			GA (Chu and Beasley, 1998)	Fix+Cuts (Osorio et al., 2002)	LP+TS (Vasquez and Hao, 2001)	Fix+LP+TS (Vasquez and Vimont, 2005)	DPSO				DPSO+LS			
n	m	α					Best	Average	t^*	T	Best	Average	t^*	T
100	5	0.25	24197	24197			24197	24197.2	0.5	10.2	24197	24197.2	0.3	68.8
100	5	0.5	43253	43253			43253	43252.9	0.5	10.1	43253	43252.9	0.2	67.7
100	5	0.75	60471	60471			60471	60471.0	0.3	9.8	60471	60471.0	0.2	61.6
100	10	0.25	22602	22602			22602	22595.8	1.7	11.1	22602	22601.9	1.7	77.5
100	10	0.5	42659	42661			**42661**	42658.2	1.4	11.2	42661	42660.6	2.6	77.0
100	10	0.75	59556	59556			59556	59554.2	0.3	11.2	59556	59555.6	0.3	68.9
100	30	0.25	21654	21656			21654	21651.0	2.4	13.0	**21660**	21658.2	4.1	86.1
100	30	0.5	41431	41437			**41435**	41430.7	2.4	13.7	**41440**	41439.9	7.2	85.1
100	30	0.75	59199	59202			59199	59196.5	1.8	14.1	59202	59201.0	3.2	78.7
250	5	0.25	60410	60413			**60414**	60407.1	8.0	26.4	**60414**	60412.3	34.5	199.0
250	5	0.5	109285	109293			**109293**	109287.2	9.9	26.3	109293	109292.8	19.2	198.3
250	5	0.75	151556	151560			**151560**	151556.5	6.2	25.6	151560	151560.3	8.5	181.2
250	10	0.25	58994	59019			**59010**	58985.8	10.6	28.4	59014	59009.5	60.0	215.2
250	10	0.5	108706	108607			**108724**	108705.7	10.1	29.1	**108727**	108722.4	49.9	204.0
250	10	0.75	151330	151363			**151339**	151329.4	8.8	28.9	151342	151339.9	24.3	196.4
250	30	0.25	56876	56959			**56893**	56855.7	11.3	32.9	56911	56891.4	55.8	218.1
250	30	0.5	106674	106686			**106682**	106654.6	13.0	34.9	**106708**	106692.4	80.2	217.5
250	30	0.75	150444	150467			**150457**	150427.6	13.6	36.4	**150471**	150461.9	55.9	206.1
500	5	0.25	120616	120610	120623	120628	**120623**	120609.8	24.3	54.5	**120630**	120623.9	174.8	440.3
500	5	0.5	219503	219504	219507	219512	**219507**	219502.5	17.0	54.5	**219513**	219510.7	138.6	422.3
500	5	0.75	302355	302361	302360	302363	**302360**	302355.1	18.1	53.4	**302362**	302360.3	82.9	401.1
500	10	0.25	118566	118584	118600	118629	**118569**	118540.8	26.4	58.0	**118605**	118580.7	242.8	467.5
500	10	0.5	217275	217297	217298	217326	**217295**	217260.6	25.6	59.4	**217312**	217288.6	216.0	459.0
500	10	0.75	302556	302562	302575	302603	**302572**	302553.4	22.1	59.5	**302591**	302577.3	165.8	434.0
500	30	0.25	115474	115520	115547	115624	**115481**	115435.6	31.0	69.8	**115559**	115493.0	208.1	454.6
500	30	0.5	216157	216180	216211	216275	**216188**	216138.2	28.8	73.8	**216221**	216184.8	189.4	452.0
500	30	0.75	302353	302373	302404	302447	**302369**	302343.7	26.4	76.7	**302405**	302382.9	193.2	438.0
Average			120153.7	120162.7			**120161.6**	120146.5	11.9	34.5	**120173.4**	120163.8	74.8	239.9

t^* = average best-solution time (CPU seconds); T = average execution time (CPU seconds).

Table 4.2 The percentage gaps between DPSO+LS and Fix+LP+LS (Vasquez and Vimont, 2005)

Problem			DPSO+LS	
n	m	α	Best	Average
500	5	0.25	-0.0019%	0.0034%
500	5	0.5	-0.0003%	0.0006%
500	5	0.75	0.0003%	0.0009%
500	10	0.25	0.0204%	0.0407%
500	10	0.5	0.0063%	0.0172%
500	10	0.75	0.0039%	0.0085%
500	30	0.25	0.0562%	0.1133%
500	30	0.5	0.0248%	0.0417%
500	30	0.75	0.0138%	0.0212%
Average			0.0137%	0.0275%

We do not compare our algorithms with Fix+LP+TS (Vasquez and Vimont, 2005), since Fix+LP+TS (Vasquez and Vimont, 2005) takes 7.6-33 hours on a Pentium4 2GHz PC to obtain the solution for each problem. In general, a metaheuristic does not perform such a long time and the solution quality of Fix+LP+TS (Vasquez and Vimont, 2005) could not be reached in a short computation time. However, we show the percentage gaps between our DPSO+LS and Fix+LP+TS (Vasquez and Vimont, 2005) in Table 4.2. The percentage gaps show that the results of our DPSO+LS very close to Fix+LP+TS (Vasquez and Vimont, 2005) in a short computation time.

4.9 Concluding Remarks

In this chapter we have presented a new discrete binary particle swarm optimization (DPSO) for solving multidimensional 0-1 knapsack problems and a local search procedure based on the concept of building blocks. The proposed DPSO adopted new concepts of particle velocity and particle movement, and obtained good solutions in a reasonable CPU time.

There are two possible extensions to this study for future research. First, the proposed DPSO can be extended for solving similar combinatorial optimization problems: for example, multidimensional 0-1 knapsack problems with generalized upper bound constraints. Second, the proposed DPSO can be extended for sequencing and scheduling problems. Similar to this chapter, we may modify the velocity based on the tabu list and the concept of building blocks, and modify particle movement based on crossover and mutation of genetic algorithm. Furthermore, we may develop the repair operator based on scheduling strategies for scheduling problems. Table 4.3 shows the summary of the DPSO for MKP.

Table 4.3 Summary of the DPSO for MKP

		Components	The concept of this components
1	Particle Position Representation	Binary variables	Since the solution of MKP is 0-1 variables, we represent the particle position by binary variables, which have the most Lamarckian.
2	Particle Velocity	Blocks	The binary variables is very appropriate to build blocks, so we implement the concept of building blocks and the relational crossover operator.
	Particle Movement	Crossover operator	
3	Decoding Operator	Repair operator based on pseudo-utility ratio	Restrict the search area in the feasible region.
4	Other Strategies	Diversification Selection Local search	The diversification strategy can prevent particles rapped in local optima. The selection strategy can further accelerate the speed of building blocks. The local search can further improve the solution quality,

4.10 Appendix

A pseudo code of the DPSO for MKP is given below:

Initialize a population of particles with random positions and velocities.

Initialize the *gbest* solution by surrogate duality approach

repeat

update velocities according to Figure 4.1.

for each particle *k* ***do***

move particle *k* according to Figure 4.2.

repair the solution of particle *k* according to repair operator (Figure 4.3).

calculate the fitness value of particle *k*.

perform local search on particle *k* (Figure 4.6).

end for

update *gbest* and *pbest* solutions according to diversification strategy (Figure 4.4).

perform selection according to selection strategy (Figure 4.5).

until maximum iterations is attained

CHAPTER 5

A PARTICLE SWARM OPTIMIZATION FOR THE JOB SHOP SCHEDULING PROBLEM

The original PSO is used to solve continuous optimization problems. Since the solution space of a shop scheduling problem is discrete, we have to modify the particle position representation, particle movement, and particle velocity to better suit PSO for scheduling problems. In the PSO for JSSP, we modified the particle position representation using preference-lists and the particle movement using a swap operator. Moreover, we propose a diversification strategy and a local search procedure for better performance.

5.1 Particle Position Representation

We implement the preference list-based representation (Davis, 1985), which has half-Lamarckian (Cheng et al., 1996). In the preference list-based representation, there is a preference list for each machine. For an n-job m-machine problem, we can represent the particle k position by an $m \times n$ matrix, and the i^{th} row is the preference list of machine i, i.e.

$$X^k = \begin{bmatrix} x_{11}^k & x_{12}^k & \cdots & x_{1n}^k \\ x_{21}^k & x_{22}^k & \cdots & x_{2n}^k \\ & \vdots & & \\ x_{m1}^k & x_{m2}^k & \cdots & x_{mn}^k \end{bmatrix}.$$

Where $x_{ij}^k \in \{1, 2, \ldots, n\}$ denotes the job on location j in the preference list of machine i. Similar to GA (Kobayashi et al., 1995) decoding a chromosome into a schedule, we also use Giffler & Thompson's heuristic (Giffler &

Thompson, 1960) to decode a particle's position to an active schedule. The G&T algorithm is shown as Figure 5.1. For example, there are 4 jobs and 4 machines as shown on $X^k = \begin{bmatrix} 1 & 2 & 4 & 3 \\ 2 & 1 & 3 & 4 \\ 2 & 3 & 4 & 1 \\ 3 & 4 & 1 & 2 \end{bmatrix}$.

Table 5.1, and the position of particle k is

$$X^k = \begin{bmatrix} 1 & 2 & 4 & 3 \\ 2 & 1 & 3 & 4 \\ 2 & 3 & 4 & 1 \\ 3 & 4 & 1 & 2 \end{bmatrix}.$$

Table 5.1 A 4╳4 job shop problem example

jobs	*machine sequence*	*processing times*
1	1, 2, 4, 3	$p_{11} = 5,\ \ p_{21} = 4,\ \ p_{41} = 2,\ \ p_{31} = 2$
2	2, 1, 3, 4	$p_{22} = 4,\ \ p_{12} = 3,\ \ p_{32} = 3,\ \ p_{42} = 2$
3	4, 1, 3, 2	$p_{43} = 2,\ \ p_{13} = 2,\ \ p_{33} = 3,\ \ p_{23} = 4$
4	3, 1, 4, 2	$p_{34} = 3,\ \ p_{14} = 2,\ \ p_{44} = 3,\ \ p_{24} = 4$

We can decode X^k to an active schedule following the G&T algorithm:

Initialization

$S = \phi$; $\Omega = \{o_{11}, o_{22}, o_{43}, o_{34}\}$.

Iteration 1

$s_{11} = 0$, $s_{22} = 0$, $s_{43} = 0$, $s_{34} = 0$; $f_{11} = 5$, $f_{22} = 4$, $f_{43} = 2$, $f_{34} = 3$; $f^* = \min\{f_{11}, f_{22}, f_{43}, f_{34}\} = 2$, $m^* = 4$.

Identify the operation set $O = \{o_{43}\}$; choose operation o_{43}, which is ahead of others in the preference list of machine 4, and add it into schedule S, as illustrated in Figure 5.2(a).

Update $\Omega = \{o_{11}, o_{22}, o_{13}, o_{34}\}$.

Notation:

o_{ij} : the operation of job j that needs to be processed on machine i.

S : the partial schedule that contains scheduled operations.

Ω : the set of schedulable operations.

s_{ij} : the earliest time at which $o_{ij} \in \Omega$ could be started.

p_{ij} : the processing time of o_{ij}.

f_{ij} : the earliest time at which $o_{ij} \in \Omega$ could be finished, $f_{ij} = s_{ij} + p_{ij}$.

G&T algorithm:

Initialize $S \leftarrow \phi$; Ω is initialized to contain all operations without predecessors.

repeat

Determine $f^* \leftarrow \min_{o_{ij} \in \Omega}\{f_{ij}\}$ and the machine m^* on which f^* could be realized.

Identify the operation set $O \leftarrow \{o_{ij} \mid s_{ij} < f^*, o_{ij} \in \Omega, i = m^*\}$.

Choose o_{ij}^* from the operation set O, where o_{ij}^* is ahead of others in the preference list of machine m^*; add o_{ij}^* to S, and assign s_{ij} as the starting time of o_{ij}^*.

Delete o_{ij}^* from Ω and include its immediate successor in Ω if o_{ij}^* is not the last operation of job j.

Until Ω is empty.

Figure 5.1 The G&T algorithm

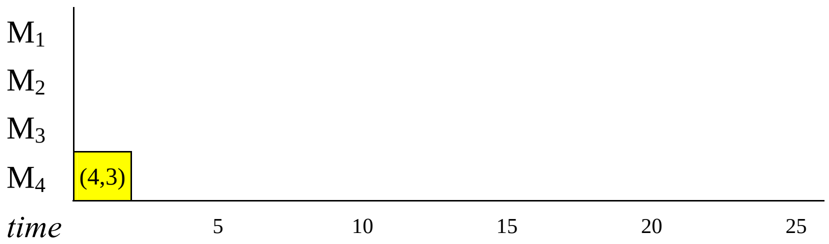


Figure 5.2(a) Partial schedule after the operation o_{43} scheduled.

Iteration 2

$s_{11}=0$, $s_{22}=0$, $s_{13}=2$, $s_{34}=0$; $f_{11}=5$, $f_{22}=4$, $f_{13}=4$, $f_{34}=3$; $f^*=\min\{f_{11}, f_{22}, f_{13}, f_{34}\}=3$, $m^*=4$.

Identify the operation set $O=\{o_{34}\}$; choose operation o_{34}, which is ahead of others in the preference list of machine 3, and add it into schedule S, as illustrated in Figure 5.2(b).

Update $\Omega=\{o_{11}, o_{22}, o_{13}, o_{14}\}$.

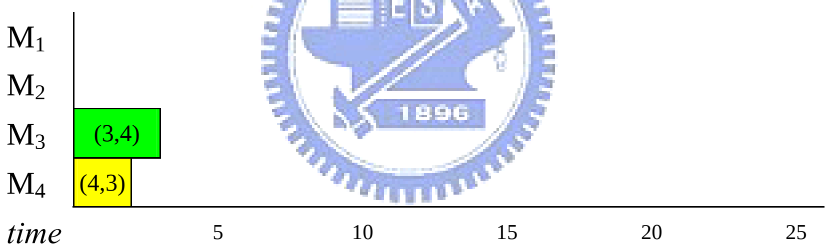


Figure 5.2(b) Partial schedule after the operation o_{34} scheduled.

Iteration 3

$s_{11}=0$, $s_{22}=0$, $s_{13}=2$, $s_{14}=3$; $f_{11}=5$, $f_{22}=4$, $f_{13}=4$, $f_{14}=5$; $f^*=\min\{f_{11}, f_{22}, f_{13}, f_{14}\}=3$, $m^*=1$.

Identify the operation set $O=\{o_{11}, o_{13}\}$; choose operation o_{11}, which is ahead of others in the preference list of machine 1, and add it into schedule S, as illustrated in Figure 5.2(c).

Update $\Omega=\{o_{21}, o_{22}, o_{13}, o_{14}\}$.

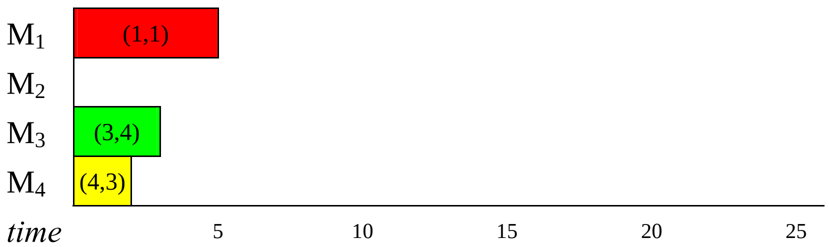


Figure 5.2(c) Partial schedule after the operation o_{11} scheduled.

Iteration 4

$s_{21}=5$, $s_{22}=0$, $s_{13}=5$, $s_{14}=5$; $f_{21}=9$, $f_{22}=4$, $f_{13}=7$, $f_{14}=7$; $f^*=\min\{f_{21}, f_{22}, f_{13}, f_{14}\}=4$, $m^*=2$.

Identify the operation set $O=\{o_{22}\}$; choose operation o_{22}, which is ahead of others in the preference list of machine 2, and add it into schedule S, as illustrated in Figure 5.2(d).

Update $\Omega=\{o_{21}, o_{12}, o_{13}, o_{14}\}$.

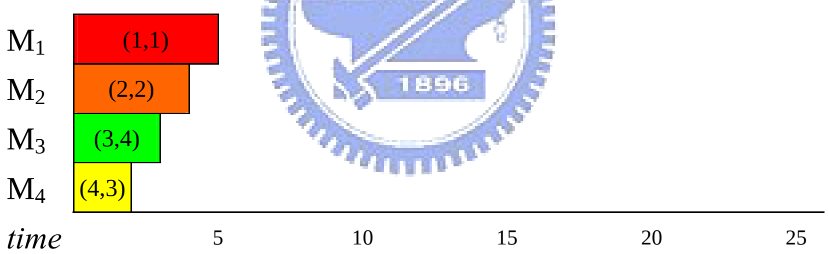


Figure 5.2(d) Partial schedule after the operation o_{22} scheduled.

Iteration 5

$s_{21}=5$, $s_{12}=5$, $s_{13}=5$, $s_{14}=5$; $f_{21}=9$, $f_{12}=7$, $f_{13}=7$, $f_{14}=7$; $f^*=\min\{f_{21}, f_{12}, f_{13}, f_{14}\}=7$, $m^*=1$.

Identify the operation set $O=\{o_{12}, o_{13}, o_{14}\}$; choose operation o_{12}, which is ahead of others in the preference list of machine 1, and add it into schedule S, as illustrated in Figure 5.2(e).

Update $\Omega=\{o_{21}, o_{32}, o_{13}, o_{14}\}$.

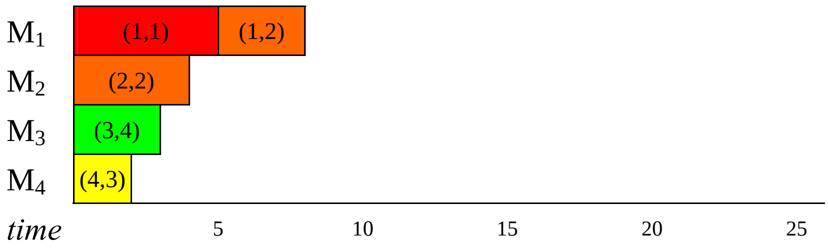


Figure 5.2(e) Partial schedule after the operation o_{12} scheduled.

Iteration 6

$s_{21}=5$, $s_{32}=8$, $s_{13}=8$, $s_{14}=8$; $f_{21}=9$, $f_{32}=11$, $f_{13}=9$, $f_{14}=9$; $f^*=\min\{f_{21}, f_{32}, f_{13}, f_{14}\}=9$, $m^*=2$.

Identify the operation set $O=\{o_{21}\}$; choose operation o_{21}, which is ahead of others in the preference list of machine 2, and add it into schedule S, as illustrated in Figure 5.2(f).

Update $\Omega=\{o_{41}, o_{32}, o_{13}, o_{14}\}$.

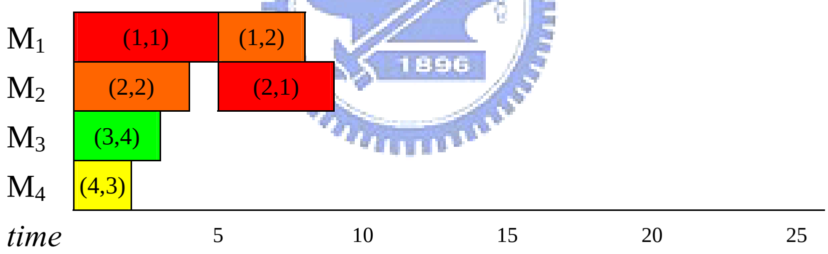


Figure 5.2(f) Partial schedule after the operation o_{21} scheduled.

Iteration 7

$s_{41}=9$, $s_{32}=8$, $s_{13}=8$, $s_{14}=8$; $f_{41}=11$, $f_{32}=11$, $f_{13}=9$, $f_{14}=9$; $f^*=\min\{f_{41}, f_{32}, f_{13}, f_{14}\}=9$, $m^*=1$.

Identify the operation set $O=\{o_{13}, o_{14}\}$; choose operation o_{14}, which is ahead of others in the preference list of machine 1, and add it into schedule S, as illustrated in Figure 5.2(g).

Update $\Omega=\{o_{41}, o_{32}, o_{13}, o_{44}\}$.

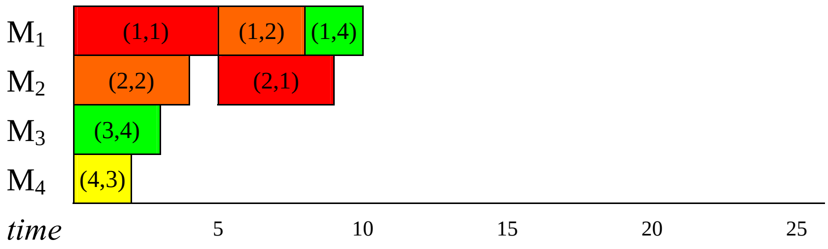


Figure 5.2(g) Partial schedule after the operation o_{14} scheduled.

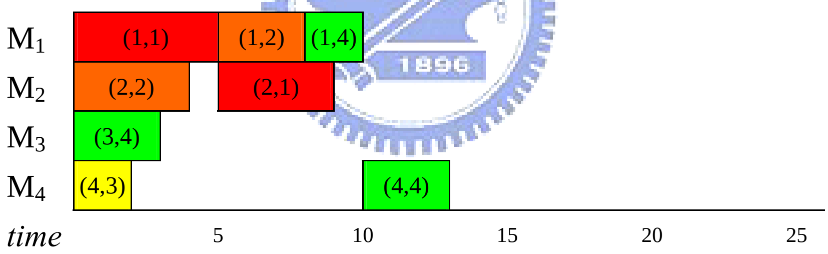
Iteration 8

s_{41} = 9, s_{32} = 8, s_{13} =10, s_{44} =10; f_{41} = 11, f_{32} = 11, f_{13} = 12, f_{44} = 13;

$f^* = \min\{ f_{41}, f_{32}, f_{13}, f_{44} \} = 11$, m^*=4.

Identify the operation set $O = \{o_{41}, o_{44}\}$; choose operation o_{44}, which is ahead of others in the preference list of machine 4, and add it into schedule S, as illustrated in Figure 5.2(h).

Update $\Omega = \{ o_{41}, o_{32}, o_{13}, o_{24} \}$.



(h) Partial schedule after the operation o_{44} scheduled.

Iteration 9

s_{41} = 13, s_{32} = 8, s_{13} =10, s_{24} =13; f_{41} = 15, f_{32} = 11, f_{13} = 12, f_{24} = 17;

$f^* = \min\{ f_{41}, f_{32}, f_{13}, f_{44} \} = 11$, m^*=3.

Identify the operation set $O = \{o_{32}\}$; choose operation o_{32}, which is ahead of others in the preference list of machine 3, and add it into schedule S, as illustrated in Figure 5.2(i).

Update $\Omega = \{ o_{41}, o_{42}, o_{13}, o_{24} \}$.

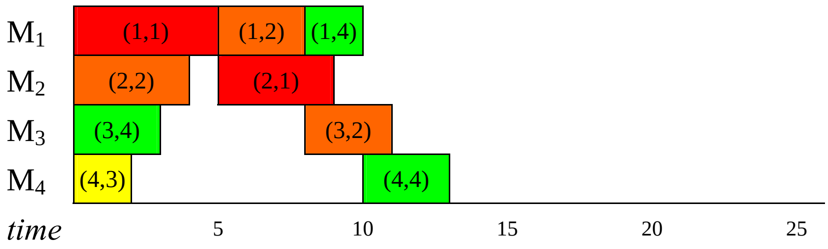


Figure 5.2(i) Partial schedule after the operation o_{32} scheduled.

Iteration 10

$s_{41} = 13$, $s_{42} = 11$, $s_{13} = 10$, $s_{24} = 13$; $f_{41} = 15$, $f_{42} = 13$, $f_{13} = 12$, $f_{24} = 17$;

$f^* = \min\{f_{41}, f_{42}, f_{13}, f_{24}\} = 12$, $m^* = 1$.

Identify the operation set $O = \{o_{13}\}$; choose operation o_{13}, which is ahead of others in the preference list of machine 1, and add it into schedule S, as illustrated in Figure 5.2(j).

Update $\Omega = \{o_{41}, o_{42}, o_{33}, o_{24}\}$.

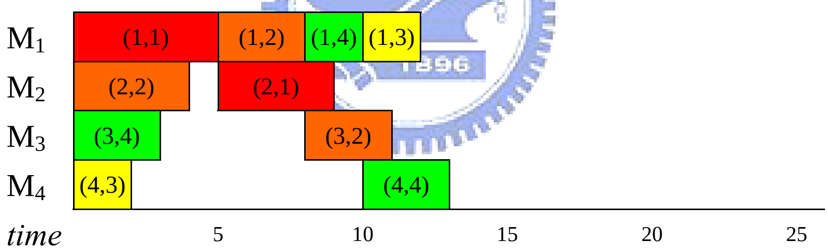


Figure 5.2(j) Partial schedule after the operation o_{13} scheduled.

Iteration 11

$s_{41} = 13$, $s_{42} = 11$, $s_{33} = 12$, $s_{24} = 13$; $f_{41} = 15$, $f_{42} = 13$, $f_{33} = 15$, $f_{24} = 17$;

$f^* = \min\{f_{41}, f_{42}, f_{33}, f_{24}\} = 13$, $m^* = 4$.

Identify the operation set $O = \{o_{42}\}$; choose operation o_{42}, which is ahead of others in the preference list of machine 4, and add it into schedule S, as illustrated in Figure 5.2(k).

Update $\Omega = \{o_{41}, o_{33}, o_{24}\}$.

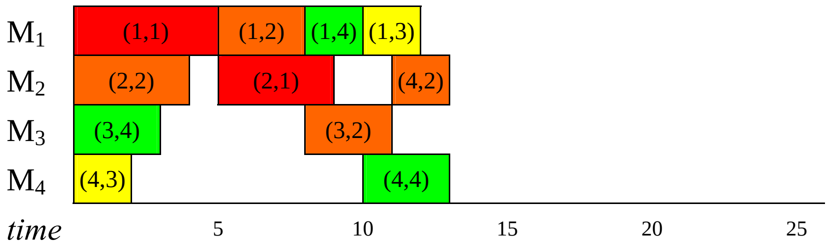


Figure 5.2(k) Partial schedule after the operation o_{42} scheduled.

Iteration 12

s_{41} = 13, s_{33} =12, s_{24} =13; f_{41} = 15, f_{33} = 15, f_{24} = 17; f^* = min{ f_{41}, f_{33}, f_{24} } = 15, m^*=4.

Identify the operation set $O = \{o_{41}\}$; choose operation o_{41}, which is ahead of others in the preference list of machine 4, and add it into schedule S, as illustrated in Figure 5.2(l).

Update $\Omega = \{ o_{31}, o_{33}, o_{24} \}$.

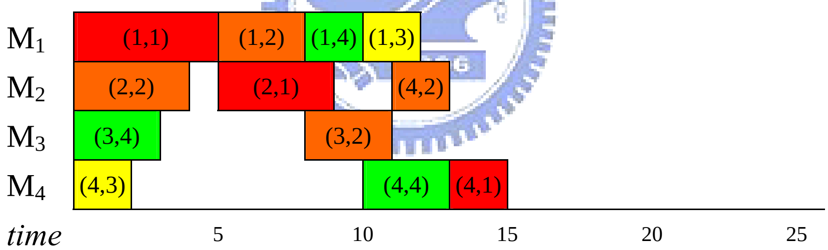


Figure 5.2(l) Partial schedule after the operation o_{41} scheduled.

Iteration 13

s_{31} = 15, s_{33} =12, s_{24} =13; f_{31} = 17, f_{33} = 15, f_{24} = 17; f^* = min{ f_{31}, f_{33}, f_{24} } = 15, m^*=3.

Identify the operation set $O = \{o_{31}, o_{33}\}$; choose operation o_{33}, which is ahead of others in the preference list of machine 3, and add it into schedule S, as illustrated in Figure 5.2(m).

Update $\Omega = \{ o_{31}, o_{23}, o_{24} \}$.

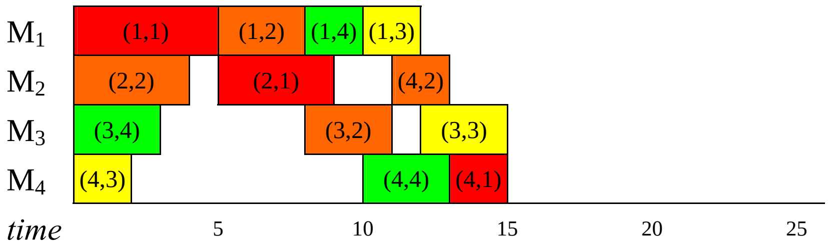


Figure 5.2(m) Partial schedule after the operation o_{33} scheduled.

Iteration 14

$s_{31}=15$, $s_{23}=15$, $s_{24}=13$; $f_{31}=17$, $f_{33}=19$, $f_{24}=17$; $f^*=\min\{f_{31}, f_{23}, f_{24}\}=17$, $m^*=3$.

Identify the operation set $O=\{o_{31}\}$; choose operation o_{31}, which is ahead of others in the preference list of machine 3, and add it into schedule S, as illustrated in (n).

Update $\Omega=\{o_{23}, o_{24}\}$.

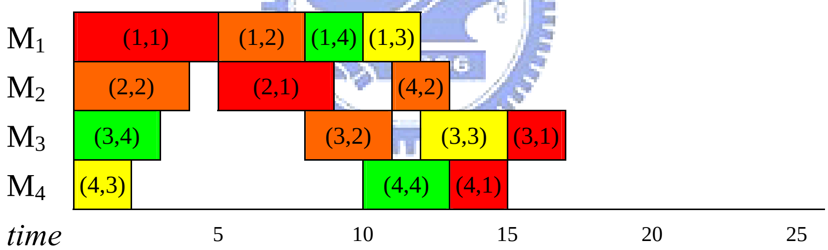


Figure 5.2(n) Partial schedule after the operation o_{31} scheduled.

Iteration 15

$s_{23}=15$, $s_{24}=13$; $f_{33}=19$, $f_{24}=17$; $f^*=\min\{f_{23}, f_{24}\}=17$, $m^*=2$.

Identify the operation set $O=\{o_{23}, o_{24}\}$; choose operation o_{23}, which is ahead of others in the preference list of machine 2, and add it into schedule S, as illustrated in (o).

Update $\Omega=\{o_{24}\}$.

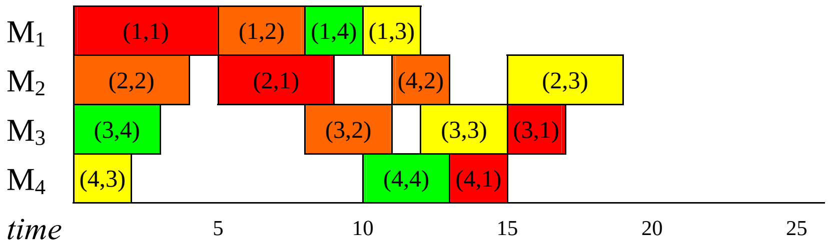


Figure 5.2(o) Partial schedule after the operation o_{23} scheduled.

Iteration 16

s_{24} =19; f_{24} = 23; f^* = min{ f_{24} } = 23, m^* =2.

Identify the operation set $O = \{o_{24}\}$; choose operation o_{24}, which is ahead of others in the preference list of machine 2, and add it into schedule S, as illustrated in Figure 5.2(p).

Update $\Omega = \phi$, and then stops.

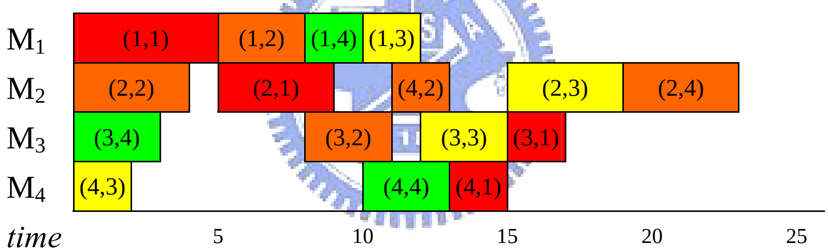


Figure 5.2(p) Partial schedule after the operation o_{24} scheduled.

Figure 5.2 An illustration of decoding a particle position into a schedule.

5.2 Particle Velocity

When a particle moves in a continuous solution space, due to inertia, the particle velocity not only moves the particle to a better position, but also prevents the particle from moving back to the current position. The velocity can be controlled by inertia weight w in equation (2.1). The larger the inertia weight, the harder the particle backs to the current position.

If we implement preference list-based representation, the velocity of operation o_{ij} of particle k is denoted by v_{ij}^k, $v_{ij}^k \in \{0, 1\}$, where o_{ij} is the operation of job j

that needs to be processed on machine i. When v_{ij}^k equals 1, it means that operation o_{ij} in the preference list of particle k (the position matrix, X^k) has just been moved to the current location, and we should not move it in this iteration. On the contrary, if operation o_{ij} is moved to a new location in this iteration, we set $v_{ij}^k \leftarrow 1$, indicating that o_{ij} has been moved in this iteration and should not been moved in the next few iterations. The particle velocity can prevent recently moved operations from moving back to the original location in the next iterations.

Just as the original PSO is applied to a continuous solution space, inertia weight w is used to control particle velocities. We randomly update velocities at the beginning of the iteration. For each particle k and operation o_{ij}, if v_{ij}^k equals 1, v_{ij}^k will be set to 0 with probability (1-w). This means that if operation o_{ij} is fixed on the current location in the preference list of particle k, o_{ij} is allowed to move in this iteration with probability (1-w). The newly moved operations will then be fixed for more iterations with larger inertia weight, and fixed for less iterations with smaller inertia weight. The pseudo code for updating velocities is given as Figure 5.3.

for each particle k and operation o_{ij} **do**
  rand ~ *U(0,1)*
  if $(v_{ij}^k \neq 0)$ *and* $(rand \geq w)$ **then**
    $v_{ij}^k \leftarrow 0$
  end if
end for

Figure 5.3 The pseudo code of updating velocities.

5.3 Particle Movement

The particle movement is based on the swap operator. If v_{ij}^k=0, the job j on x_i^k will be moved to the corresponding location of $pbest_i^k$ with probability c_1, and will be moved to the corresponding location of $gbest_i$ with probability c_2. Where x_i^k is the preference list of machine *i* of particle *k*, $pbest_i^k$ is the preference list of machine *i* of *k*th $pbest$ solution, $gbest_i$ is the preference list of machine *i* of $gbest$ solution, c_1 and c_2 are constant between 0 and 1, and $c_1 + c_2 \leq 1$. The process is described as follows:

Step 1: Randomly choose a location l in x_i^k.

Step 2: Denote the job on location l in x_i^k by J_1.

Step 3: Find out the location of J_1 in $pbest_i^k$ with probability c_1, or find out the location of J_1 in $gbest_i$ with probability c_2. Denote the location that has been found in $pbest_i^k$ or $gbest_i$ by l', and denote the job in location l' in x_i^k by J_2.

Step 4: If J_2 has been denoted, $v_{iJ_1}^k$=0, and $v_{iJ_2}^k$=0, then swap J_1 and J_2 in x_i^k, and set $v_{iJ_1}^k \leftarrow 1$.

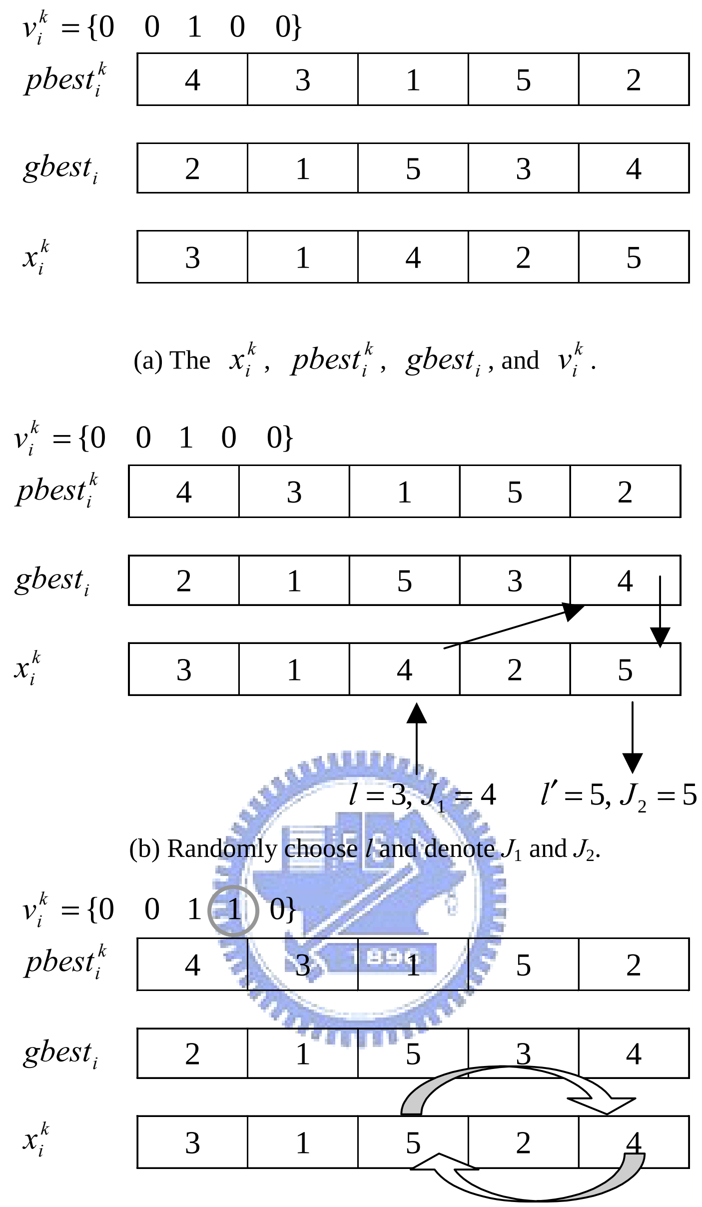
Step 5: If all the locations in x_i^k have been considered, then stop. Otherwise, if $l < n$, then set $l \leftarrow l+1$, else $l \leftarrow 1$, and go to Step 2, where *n* is the number of jobs.

For example, there is a 5-job problem, and x_i^k, $pbest_i^k$, $gbest_i$, and v_i^k are showed as Figure 5.4(a). We set c_1=0.5 and c_2=0.3 in this instance.

In Step 1, we randomly choose a location l=3. In Step 2, the job in the 3rd location in x_i^k is job 4, i.e. J_1= 4. In Step 3, we generate a random variable *rand* between 0 and 1, and the generated random variable *rand* is 0.6. Since $c_1 < rand \leq c_1 + c_2$, we find out the location of J_1 in $gbest_i$. The location l'=5, and the job in the 5th location in x_i^k is job 5, i.e. J_2=5. Step 1 to Step 3 is shown as

Figure 5.4(b). In Step 4, since $v_{i4}^k = 0$ and $v_{i5}^k = 0$, swap job 4 and job 5 in x_i^k and set $v_{i4}^k \leftarrow 1$ is shown as Figure 5.4(c). In Step 5, set $l \leftarrow 4$, and go to Step 2. Repeat the procedure until all the locations in x_i^k have been considered.

We also adopt a mutation operator in our algorithm. After a particle moves to a new position, we randomly choose a machine and two jobs on the machine, and then swap these two jobs, disregarding v_{ij}^k. The particle movement pseudo code is given as Figure 5.5.



(a) The x_i^k, $pbest_i^k$, $gbest_i$, and v_i^k.

(b) Randomly choose l and denote J_1 and J_2.

(c) Swap job4 and job5 in x_i^k; set $v_{i4}^k \leftarrow 1$.

Figure 5.4 An instance of particle movement.

for $i \leftarrow 1$ ***to*** m ***do*** //for machine 1 to machine m

$\quad l_{start} \leftarrow$ *an integer random number between* 1 *to* n

$\quad l \leftarrow l_{start}$

$\quad$ ***for*** $j \leftarrow 1$ ***to*** n ***do*** //for all location

$\quad\quad rand \sim U(0,1)$

$\quad\quad$ ***if*** $(rand \le c_1)$ ***then***

$\quad\quad\quad J_1 \leftarrow x_{il}^k$

$\quad\quad\quad l' \leftarrow$ *the location of* J_1 *in* $pbest_i^k$

$\quad\quad\quad J_2 \leftarrow x_{il'}^k$

$\quad\quad\quad$ ***if*** $(v_{iJ_1}^k = 0)$ ***and*** $(v_{iJ_2}^k = 0)$ ***and*** $(J_1 \ne J_2)$ ***then***

$\quad\quad\quad\quad x_{il}^k \leftarrow J_2;\ \ x_{il'}^k \leftarrow J_1;\ \ v_{iJ_1}^k \leftarrow 1$

$\quad\quad\quad$ ***end if***

$\quad\quad$ ***end if***

$\quad\quad$ ***if*** $(c_1 < rand \le c_1 + c_2)$ ***then***

$\quad\quad\quad J_1 \leftarrow x_{il}^k$

$\quad\quad\quad l' \leftarrow$ *the location of* J_1 *in* $gbest_i$

$\quad\quad\quad J_2 \leftarrow x_{il'}^k$

$\quad\quad\quad$ ***if*** $(v_{iJ_1}^k = 0)$ ***and*** $(v_{iJ_2}^k = 0)$ ***and*** $(J_1 \ne J_2)$ ***then***

$\quad\quad\quad\quad x_{il}^k \leftarrow J_2;\ \ x_{il'}^k \leftarrow J_1;\ \ v_{iJ_1}^k \leftarrow 1$

$\quad\quad\quad$ ***end if***

$\quad\quad$ ***end if***

$\quad\quad l \leftarrow l_{start} + j$

$\quad\quad$ ***if*** $(l > n)$ ***then***

$\quad\quad\quad l \leftarrow l - n$

$\quad\quad$ ***end if***

$\quad$ ***end for***

end for

//mutation operator

$M \leftarrow$ *randomly choose a machine between* 1 *to m*

$l \leftarrow$ *randomly choose a location between* 1 *to* n

$l' \leftarrow$ *randomly choose a location between* 1 *to* n

$J_1 \leftarrow x_{Ml}^k;\ \ J_2 \leftarrow x_{Ml'}^k$

$x_{Ml}^k \leftarrow J_2;\ \ x_{Ml'}^k \leftarrow J_1$

$v_{MJ_1}^k \leftarrow 1;\ \ v_{MJ_2}^k \leftarrow 1$

//mutation operator

Figure 5.5 Pseudo code of particle movement.

5.4 The Diversification Strategy

If all the particles have the same *pbest* solutions, they will be trapped into local optima. To prevent such a situation, we proposed a diversification strategy to keep the *pbest* solutions different (i.e. keeps the makespans of *pbest* solutions different). In the diversification strategy, the *pbest* solution of each particle is not the best solution found by the particle itself, but one of the best *N* solutions found by the swarm so far where *N* is the size of the swarm. Once any particle generates a new solution, the *pbest* and *gbest* solutions will be updated in these three situations:

1. If the particle's fitness value is better than the fitness value of the *gbest* solution, set the worst *pbest* solution equal to the current *gbest* solution, and set the *gbest* solution equal to the particle solution.

2. If the particle's fitness value is worse than the *gbest* solution, but better then the worst *pbest* solution and not equal to any *gbest* or *pbest* solution, set the worst *pbest* solution equal to the particle solution.

3. If the particle's fitness value is equal to any *pbest* or *gbest* solution, replace the *pbest* or *gbest* solution (whose fitness value is equal to the particle fitness value) with the particle solution.

The pseudo code for updating the *pbest* solution and *gbest* solution with diversification strategy is given as Figure 5.6.

N: the size of the swarm

S^k : the schedule generated by particle k

$pbest^{worst}$: the worst solution of pbest solutions

$C_{\max}(S^k)$: the makespan of S^k

// ↓ situation 1 as described in 3.4

if $(C_{\max}(S^k) < C_{\max}(gbest))$ ***then***

  $pbest^{worst} \leftarrow gbest$; $gbest \leftarrow S^k$

// ↑ situation 1 as described in 3.4

else if $(C_{\max}(S^k) \le C_{\max}(pbest^{worst}))$ ***then***

  // ↓ situation 3 as described in 3.4

  $the\_same = 0$

  if $(C_{\max}(S^k) = C_{\max}(gbest))$ ***then***

    $gbest \leftarrow S^k$; $the\_same = 1$

  else

    for $k' \leftarrow 1$ ***to*** N ***do***

      if $(C_{\max}(S^k) = C_{\max}(pbest^{k'}))$ ***then***

        $pbest^{k'} \leftarrow S^k$; $the\_same = 1$

        break

      end if

    end for

  end if

  // ↑ situation 3 as described in 3.4

  // ↓ situation 2 as described in 3.4

  if $(the\_same = 0)$ ***then***

    $pbest^{worst} \leftarrow S^k$

  end if

  // ↑ situation 2 as described in 3.4

end if

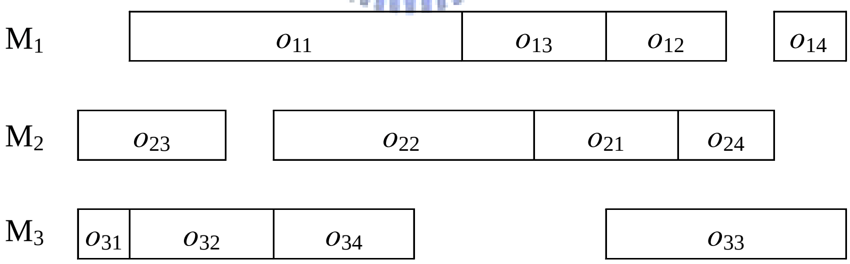
Figure 5.6 Pseudo code of updating *pbest* solution and *gbest* solution with diversification strategy.

5.5 Local Search

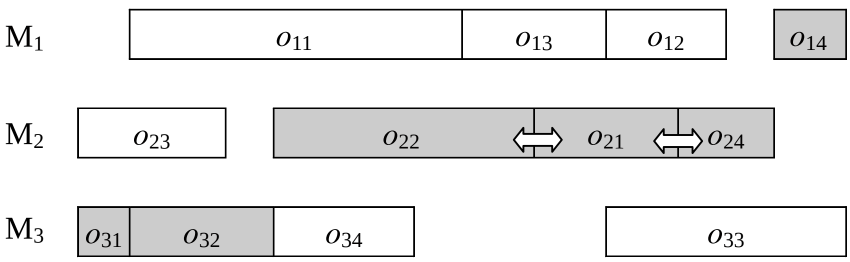
The tabu search is a metaheuristic approach and a strong local search mechanism. In the tabu search, the algorithm starts from an initial solution and improves it iteratively to find a near-optimal solution. This method was proposed and formalized primarily by Glover (1986, 1989, 1990). We applied the tabu search proposed by Nowicki and Smutnicki (1996) but without back jump tracking. We briefly describe Nowicki and Smutnicki's method as follows:

. ***The neighborhood structure***

Nowicki and Smutnicki's method randomly chooses a critical path in the current schedule, and then represents the critical path in terms of blocks. The neighborhood exchanges the first two and the last two operations in every block, but excludes the first and last operations in the critical path. The research of Jain et al. (2000) shows that the strategy used to generate the critical path does not materially affect the final solution. Therefore, in this research, we randomly choose one critical path if there is more than one critical path. For example, there is a schedule for a 4-job, 3-machine problem, as shown in Figure 5.7(a). We can find that there are two critical paths: $CP_1=\{o_{31}, o_{11}, o_{13}, o_{33}\}$ and $CP_2=\{o_{31}, o_{32}, o_{22}, o_{21}, o_{24}, o_{14}\}$, where o_{ij} is the operation of job j that needs to be processed on machine i. If we randomly choose CP_2, we can represent CP_2 in terms of blocks: $\{o_{31}, o_{32}\}$, $\{o_{22}, o_{21}, o_{24}\}$, and $\{o_{14}\}$. The possible moves in this schedule are exchanging $\{o_{22}, o_{21}\}$ or $\{o_{21}, o_{24}\}$ (see Figure 5.7(b)).



(a) An instance of job shop schedule.



(b) Neighborhood defined by Nowicki & Smutnicki (1996).

Figure 5.7 An illustration of neighborhoods in tabu search.

. ***Tabu list***

The tabu list consists of *maxt* operation pairs that have been moved in the last *maxt* moves in the tabu search. If a move $\{o_{iJ_1},\ o_{iJ_2}\}$ has been performed, this move replaces the oldest move in the tabu list, and moving these same two operations is not permitted while the move is recorded in the tabu list.

. ***Back jump tracking***

When finding a new best solution, store the current state (the new best solution, set of moves, and tabu list) in a list L. After the tabu search algorithm performs *maxiter_tabu* iterations, restart the tabu search algorithm from the latest recorded state, and repeat it until the list L is empty. We did not implement the back jump tracking in our algorithm to reduce computation time.

We implement a tabu search procedure after a particle generates a new solution for further improved solution quality. The tabu search will be stopped after 100 moves that do not improve the solution. The research of Jain et al. (2000) shows that the solution quality of tabu search (Nowicki & Smutnicki, 1996) is mainly affected by its initial solution. Therefore, in the hybrid PSO, the purpose of the PSO process is to provide good and diverse initial solutions to the tabu search.

5.6 Computational Results

There are three PSOs we tested: (1) priority-based PSO, of which the particle position is represented by the priorities of operations, and implements the original PSO design; (2) preference list-based PSO, of which the particle position is represented by a preference list of machines; (3) hybrid PSO (HPSO), which is the preference list-based PSO with a local search mechanism. The PSOs were tested on Fisher and Thompson (1963) (FT06, FT10 and FT20), Lawrence (1984) (LA01 to LA40) and Taillard (1993) (TA01 to TA80) test problems. These problems are available on the OR-Library web site (Beasley, 1990) (URL: http://people.brunel.ac.uk/~mastjjb/jeb/info.html) and Taillard's web site (URL: http://ina2.eivd.ch/Collaborateurs/etd/problemes.dir/ordonnancement.dir/ordonnancement.html).

In the preliminary experiment, four swarm sizes *N* (10, 20, 30, 50) were tested, where *N*=30 was superior and used for all further studies. The other parameters of the priority-based PSO were set to the same common settings as most of the previous research: $c_1 = 2.0$, $c_2 = 2.0$, the inertia weight w is decreased linearly from 0.9 to 0.4 during a run, and the maximum value of $|x_{ij}|$ and $|v_{ij}|$, *Xmax* and *Vmax* are equal to the number of jobs *n* and *n*/5 respectively.

The parameters of the preference list-based PSO are determined experimentally. The parameters c_1 and c_2 were tested between 0.1 and 0.5 in increments of 0.1, and the parameter w was tested between 0 and 0.9 in increments of 0.1. The settings $c_1 = 0.5$, $c_2 = 0.3$ and $w = 0.5$ were superior. The length of the tabu list *maxt* was set to 8 where the value is derived from Nowicki and Smutnicki (1996). The tabu search will be stopped after 100 moves that do not improve the solution. The priority-based PSO and the preference list-based PSO will be terminated after 10^5 iterations, and HPSO will be terminated after 10^3 iterations. The number of iterations is determined by the computation time compared with Pezzella and Merelli (2000) and Gonçalves et al. (2005).

The program was coded in Visual C++, optimized by speed, and run on an AMD

Athlon 1700+ PC twenty times for each of the 123 problems. The proposed algorithm is compared with Shifting Bottleneck (Adams et al., 1988; Balas & Vazacopoulos, 1998), Tabu Search (Sun et al., 1995; Nowicki & Smutnicki, 1996; Pezzella & Merelli, 2000), and Genetic Algorithm (Wang & Zheng, 2001; Gonçalves et al., 2005).

The computational results of FT and LA test problems are shown as Table 5.2. The results show that the preference list-based PSO we proposed is much better than the original design, the priority-based PSO. Since the number of instances tested by each method is different, we cannot compare the result by average gap directly. Nevertheless, the result obtained by HPSO is better then other algorithms that tested all of the 43 instances, and the HPSO obtained the best-known solution for 41 of the 43 instances.

Table 5.3 shows the average computation time on FT and LA test problems in CPU seconds. The ‘best-solution time’ is the average time that the algorithm takes to first reach the final best solution, and the ‘total time’ is the average total computation time that the algorithm takes during a run. In HPSO, there is about 99% computation time spent on local search process. As mentioned in section 5.5, the solution quality of tabu search (Nowicki & Smutnicki, 1996) is mainly affected by its initial solution, and the main purpose of the PSO process is to provide good and diverse initial solutions to tabu search. Therefore, the computational results show that the hybrid method, HPSO, performs better than both TSAB and PSO, and its average gap is 0.356% less than PSO.

We further tested HPSO on TA test problems (Taillard, 1993). The computational results are shown in Table 5.4, and we particularly compared HPSO with TSSB (Pezzella & Merelli, 2000) in Table 5.5. Since the maximum computation time of TSSB is about 3×10^4 seconds and our machine is about ten times faster then TSSB

(Pezzella & Merelli, 2000), we limited the maximum computation time of HPSO in 3 $\times 10^3$ seconds. As mentioned above, 99% of the computation time is spent on the local search process in HPSO. Therefore, we do not reduce the computation time by decreasing the number of iterations, but decreasing the percentage of particles that perform a local search procedure. The HPSO will also be terminated after 10^3 iterations, but there are only 34.6% of particles randomly chosen to perform the local search procedure in each iteration on TA51 to TA60 test problems, 26.6% on TA61 to TA70 test problems, and 6.4% on TA71 to TA80 test problems.

Table 5.5 shows the comparison with TSSB (Pezzella & Merelli, 2000). The HPSO performs better than TSSB on 7 of 8 problem sizes, and only worse than TSSB on the 100$\times$20 problem size. In the 100$\times$20 problem sizes, the final best solutions are obtained after 890 iterations of the average (so the best-solution time is very close to the total time). Since the HPSO only performs 10^3 iterations for each run, it shows that the particles of HPSO did not converge in 10^3 iterations, and can further improve the solutions by increasing the maximum iteration. However, since we want to compare HPSO with TSSB, we do not consider increasing the maximum iteration because it takes too much computation time.

Table 5.2 Computational result of FT and LA test problems.

Problem	Size (n×m)	Best Known Solution (BKS)	Shifting Bottleneck				Tabu Search		
			SBI	SBII	SB-RGLS1	SB-RGLS2	ACM	TSAB	TSSB
			Adams et al. (1988)		Balas & Vazacopoulos (1998)		Sun et al. (1995)	Nowicki & Smutnicki (1996)	Pezzella & Merelli (2000)
FT06	6×6	55	**55**	**55**	–	–	–	**55**	**55**
FT10	10×10	930	1015	**930**	**930**	**930**	**930**	**930**	**930**
FT20	20×5	1165	1290	1178	–	–	–	**1165**	**1165**
LA01	10×5	666	**666**	**666**	–	–	–	**666**	**666**
LA02	10×5	655	720	669	**655**	**655**	–	**655**	**655**
LA03	10×5	597	623	605	–	–	–	**597**	**597**
LA04	10×5	590	597	593	–	–	–	**590**	**590**
LA05	10×5	593	**593**	**593**	–	–	–	**593**	**593**
LA06	15×5	926	**926**	**926**	–	–	–	**926**	**926**
LA07	15×5	890	**890**	**890**	–	–	–	**890**	**890**
LA08	15×5	863	868	**863**	–	–	–	**863**	**863**
LA09	15×5	951	**951**	**951**	–	–	–	**951**	**951**
LA10	15×5	958	959	959	–	–	–	**958**	**958**
LA11	20×5	1222	**1222**	**1222**	–	–	–	**1222**	**1222**
LA12	20×5	1039	**1039**	**1039**	–	–	–	**1039**	**1039**
LA13	20×5	1150	**1150**	**1150**	–	–	–	**1150**	**1150**
LA14	20×5	1292	**1292**	**1292**	–	–	–	**1292**	**1292**
LA15	20×5	1207	**1207**	**1207**	–	–	–	**1207**	**1207**
LA16	10×10	945	1021	978	–	–	975	**945**	**945**
LA17	10×10	784	796	787	–	–	**784**	**784**	**784**
LA18	10×10	848	891	859	–	–	**848**	**848**	**848**
LA19	10×10	842	875	860	**842**	**842**	**842**	**842**	**842**
LA20	10×10	902	924	914	–	–	**902**	**902**	**902**
LA21	15×10	1046	1172	1084	1048	**1046**	1074	1047	**1046**
LA22	15×10	927	1040	944	–	–	941	**927**	**927**
LA23	15×10	1032	1061	**1032**	–	–	**1032**	**1032**	**1032**
LA24	15×10	935	1000	976	937	**935**	954	939	938
LA25	15×10	977	1048	1017	**977**	**977**	1010	**977**	979
LA26	20×10	1218	1304	1224	–	–	**1218**	**1218**	**1218**
LA27	20×10	1235	1325	1291	**1235**	**1235**	1277	1236	**1235**
LA28	20×10	1216	1256	1250	–	–	1245	**1216**	**1216**
LA29	20×10	1157	1294	1239	1164	1164	1234	1160	1168
LA30	20×10	1355	1403	**1355**	–	–	**1355**	**1355**	**1355**
LA31	30×10	1784	**1784**	**1784**	–	–	**1784**	**1784**	**1784**
LA32	30×10	1850	**1850**	**1850**	–	–	**1850**	**1850**	**1850**
LA33	30×10	1719	**1719**	**1719**	–	–	**1719**	**1719**	**1719**
LA34	30×10	1721	**1721**	**1721**	–	–	**1721**	**1721**	**1721**
LA35	30×10	1888	**1888**	**1888**	–	–	**1888**	**1888**	**1888**
LA36	15×15	1268	1351	1305	**1268**	**1268**	1303	**1268**	**1268**
LA37	15×15	1397	1485	1423	**1397**	**1397**	1422	1407	1411
LA38	15×15	1196	1280	1255	1198	**1196**	1245	**1196**	1201
LA39	15×15	1233	1321	1273	**1233**	**1233**	1269	**1233**	1240
LA40	15×15	1222	1326	1269	1226	1224	1255	1229	1233
Average Gap			3.8796%	1.3838%	0.1157%	0.0591%	1.5184%	0.0501%	0.1015%
# of instance			43	43	13	13	26	43	43
# of BKS obtained			16	20	8	11	13	37	36

Table 5.2 (Continued)

Genetic Algorithm		Particle Swarm Optimization					
GASA	HGA-Param	PSO-priority based		PSO-permutation based		HPSO	
Wang & Zheng (2001)	Gonçalves et al. (2005)	Best solution	Average	Best solution	Average	Best solution	Average
55	**55**	**55**	58.9	**55**	**55.0**	**55**	**55.0**
930	**930**	1007	1086.0	937	965.2	**930**	932.0
1165	**1165**	1242	1296.7	**1165**	1178.8	**1165**	**1165.0**
666	**666**	681	705.0	**666**	**666.0**	**666**	**666.0**
–	**655**	694	729.7	**655**	662.1	**655**	**655.0**
–	**597**	633	657.5	**597**	602.3	**597**	**597.0**
–	**590**	611	648.1	**590**	592.9	**590**	**590.0**
–	**593**	**593**	601.1	**593**	**593.0**	**593**	**593.0**
926	**926**	**926**	940.2	**926**	**926.0**	**926**	**926.0**
–	**890**	**890**	941.0	**890**	**890.0**	**890**	**890.0**
–	**863**	**863**	896.6	**863**	**863.0**	**863**	**863.0**
–	**951**	953	991.8	**951**	**951.0**	**951**	**951.0**
–	**958**	**958**	976.1	**958**	**958.0**	**958**	**958.0**
1222	**1222**	**1222**	1235.3	**1222**	**1222.0**	**1222**	**1222.0**
–	**1039**	**1039**	1058.4	**1039**	**1039.0**	**1039**	**1039.0**
–	**1150**	**1150**	1179.0	**1150**	**1150.0**	**1150**	**1150.0**
–	**1292**	**1292**	1292.2	**1292**	**1292.0**	**1292**	**1292.0**
–	**1207**	1232	1271.7	**1207**	**1207.0**	**1207**	**1207.0**
945	**945**	1006	1033.5	**945**	969.8	**945**	945.2
–	**784**	833	883.5	**784**	787.1	**784**	**784.0**
–	**848**	901	959.9	**848**	856.8	**848**	**848.0**
–	**842**	895	945.8	**842**	851.5	**842**	**842.0**
–	907	963	1014.0	907	913.3	**902**	902.3
1058	**1046**	1201	1247.5	1055	1085.5	**1046**	1049.8
–	935	1046	1142.5	935	950.5	**927**	**927.0**
–	**1032**	1146	1205.1	**1032**	**1032.0**	**1032**	**1032.0**
–	953	1082	1140.9	937	967.8	**935**	937.9
–	986	1107	1176.6	983	1005.9	**977**	978.2
1218	**1218**	1409	1468.0	**1218**	1219.7	**1218**	**1218.0**
–	1256	1437	1495.4	1252	1269.1	**1235**	1251.4
–	1232	1434	1487.4	**1216**	1241.7	**1216**	**1216.0**
–	1196	1359	1429.8	1179	1215.8	1163	1168.8
–	**1355**	1517	1557.0	**1355**	**1355.0**	**1355**	**1355.0**
1784	**1784**	1886	1942.5	**1784**	**1784.0**	**1784**	**1784.0**
–	**1850**	2000	2065.6	**1850**	**1850.0**	**1850**	**1850.0**
–	**1719**	1832	1896.8	**1719**	**1719.0**	**1719**	**1719.0**
–	**1721**	1876	1953.5	**1721**	**1721.0**	**1721**	**1721.0**
–	**1888**	2027	2074.5	**1888**	**1888.0**	**1888**	**1888.0**
1292	1279	1437	1541.0	1291	1317.5	**1268**	1271.3
–	1408	1539	1628.0	1442	1475.1	**1397**	1401.6
–	1219	1370	1445.1	1228	1251.1	**1196**	1200.5
–	1246	1436	1499.4	**1233**	1285.6	**1233**	**1233.0**
–	1241	1380	1457.4	1236	1258.0	1224	1226.2
0.2764%	0.3916%	7.4021%	12.0940%	0.3719%	1.3491%	0.0159%	0.1091%
11	43	43		43		43	
9	31	10		31		41	

Table 5.3 Computation time of FT and LA test problems (in CPU seconds).

		HGA-Param	Particle Swarm Optimization**					
		Gonçalves et al. (2005)*	PSO-priority based		PSO-permutation based		HPSO	
Problem	Size (n×m)	Total time	Best solution time	Total time	Best solution time	Total time	Best solution time	Total time
FT06	6×6	13	0.0	34	0.0	32	0.0	28
FT10	10×10	292	1.0	112	21.7	91	4.1	157
FT20	20×5	204	3.0	180	19.2	138	19.8	219
LA01-05	10×5	40	0.4	60	5.3	50	0.5	38
LA06-10	15×5	94	1.0	114	0.1	92	0.1	61
LA11-15	20×5	192	3.5	177	0.5	143	0.1	100
LA16-20	10×10	227	0.6	109	15.5	90	19.9	139
LA21-25	15×10	602	4.8	208	37.2	164	59.6	295
LA26-30	20×10	1303	12.6	325	103.1	259	90.5	579
LA31-35	30×10	3691	46.9	652	31.4	520	3.0	1462
LA36-40	15×15	1920	7.4	331	68.4	254	105.2	471

* Run on an AMD Thunderbird 1.333 GHz PC.
**Run on an AMD Athlon 1700+ PC.

Table 5.4 Computational result of TA test problems.

Problem	Size (n×m)	Optimal solution (or upper bound)	TSAB	TSSB	HPSO	
			Nowicki & Smutnicki (1996)	Pezzella & Merelli (2000)	Best solution	Average
TA01	15×15	1231		1241	**1231**	1236
TA02	15×15	1244	**1244**	**1244**	**1244**	1245
TA03	15×15	1218	1222	1222	**1218**	1224
TA04	15×15	1175		**1175**	**1175**	1180
TA05	15×15	1224	1233	1229	**1224**	1233
TA06	15×15	1238		1245	**1238**	1248
TA07	15×15	1227		1228	1228	1229
TA08	15×15	1217	1220	1220	**1217**	1220
TA09	15×15	1274	1282	1291	**1274**	1283
TA10	15×15	1241	1259	1250	1249	1264
TA11	20×15	(1359)		1371	1366	1386
TA12	20×15	(1367)	1377	1379	1370	1380
TA13	20×15	(1342)		1362	1350	1364
TA14	20×15	1345	**1345**	**1345**	**1345**	1350
TA15	20×15	(1339)		1360	1350	1364
TA16	20×15	(1360)		1370	1368	1377
TA17	20×15	1462		1481	1473	1480
TA18	20×15	(1396)	1413	1426	1407	1425
TA19	20×15	(1335)	1352	1351	**1335**	1353
TA20	20×15	(1348)	1362	1366	1358	1373
TA21	20×20	(1644)		1659	1658	1679
TA22	20×20	(1600)		1623	1614	1625
TA23	20×20	(1557)		1573	1559	1578
TA24	20×20	(1646)		1659	1654	1664
TA25	20×20	(1595)		1606	1616	1632
TA26	20×20	(1645)	1657	1666	1662	1679
TA27	20×20	(1680)		1697	1690	1712
TA28	20×20	(1603)		1622	1617	1627
TA29	20×20	(1625)	1629	1635	1634	1645
TA30	20×20	(1584)		1614	1589	1613
TA31	30×15	1764	1766	1771	1766	1772
TA32	30×15	(1795)	1841	1840	1823	1848
TA33	30×15	(1791)	1832	1833	1818	1834
TA34	30×15	(1829)		1846	1844	1879
TA35	30×15	2007		**2007**	**2007**	2010
TA36	30×15	1819		1825	1825	1843
TA37	30×15	1771	1815	1813	1795	1808
TA38	30×15	1673	1700	1697	1681	1701
TA39	30×15	1795	1811	1815	1796	1810
TA40	30×15	(1674)	1720	1725	1698	1714

Table 5.4 (Continued)

Problem	Size (n×m)	Optimal solution (or upper bound)	TSAB	TSSB	HPSO	
			Nowicki & Smutnicki (1996)	Pezzella & Merelli (2000)	Best solution	Average
TA41	30×20	(2018)		2045	2047	2071
TA42	30×20	(1949)		1979	1970	1984
TA43	30×20	(1858)		1898	1899	1928
TA44	30×20	(1983)		2036	2019	2039
TA45	30×20	(2000)		2021	2010	2032
TA46	30×20	(2015)		2047	2041	2070
TA47	30×20	(1903)		1938	1935	1958
TA48	30×20	(1949)	2001	1996	1994	2022
TA49	30×20	(1967)		2013	1992	2015
TA50	30×20	(1926)		1975	1975	1998
TA51	50×15	2760		**2760**	**2760**	2760
TA52	50×15	2756		**2756**	**2756**	2758
TA53	50×15	2717		**2717**	**2717**	2717
TA54	50×15	2839		**2839**	**2839**	2840
TA55	50×15	2679	**2679**	2684	**2679**	2694
TA56	50×15	2781		**2781**	**2781**	2785
TA57	50×15	2943		**2943**	**2943**	2943
TA58	50×15	2885		**2885**	**2885**	2885
TA59	50×15	2655		**2655**	**2655**	2666
TA60	50×15	2723		**2723**	**2723**	2732
TA61	50×20	2868	**2868**	**2868**	**2868**	2896
TA62	50×20	2869	2902	2942	2930	2958
TA63	50×20	2755	**2755**	**2755**	**2755**	2774
TA64	50×20	2702	**2702**	**2702**	**2702**	2718
TA65	50×20	2725	**2725**	**2725**	2735	2759
TA66	50×20	2845	**2845**	**2845**	2848	2869
TA67	50×20	2825	2841	2865	2840	2861
TA68	50×20	2784	**2784**	**2784**	**2784**	2802
TA69	50×20	3071	**3071**	**3071**	**3071**	3096
TA70	50×20	2995	**2995**	**2995**	3005	3041
TA71	100×20	5464		**5464**	5519	5595
TA72	100×20	5181		**5181**	5211	5305
TA73	100×20	5568		**5568**	5581	5655
TA74	100×20	5339		**5339**	5355	5412
TA75	100×20	5392		**5392**	5466	5563
TA76	100×20	5342		**5342**	5396	5504
TA77	100×20	5436		**5436**	5444	5493
TA78	100×20	5394		**5394**	**5394**	5476
TA79	100×20	5358		**5358**	5363	5434
TA80	100×20	5183	**5183**	**5183**	5209	5364
Average Gap			0.7792%	0.8122%	0.5659%	1.4651%
# of instance			33	80	80	
# of BKS obtained			12	31	27	

Table 5.5 Comparison with TSSB (Pezzella & Merelli, 2000) on TA test problems.

Problem	Size (n×m)	TSSB*		HPSO**		
		Average gap	Total time	Average gap	Time to get best solution	Total time
TA01-10	15×15	0.4502%	2175	0.0726%	99	514
TA11-20	20×15	1.1537%	2526	0.5023%	345	855
TA21-30	20×20	1.0840%	34910	0.7029%	401	1238
TA31-40	30×15	1.4475%	14133	0.7654%	1185	2026
TA41-50	30×20	1.9474%	11512	1.6133%	1734	2769
TA51-60	50×15	0.0187%	421	0.0000%	565	2909***
TA61-70	50×20	0.3960%	6342	0.3463%	2322	2862***
TA71-80	100×20	0.0000%	231	0.5244%	2797	3137***
Total Average Gap		0.8122%		0.5659%		

* Run on a Pentium 133 MHz PC.
**Run on an AMD Athlon 1700+ PC.
***Decreasing the percentage to perform local search procedure reduces the computation time.

5.7 Concluding Remarks

We have presented a PSO for the job shop scheduling problem. We modified the representation of particle position, particle movement, and particle velocity to better suit it for JSSP. We also applied Tabu Search to improve solution quality. The computational results show that HPSO can obtain better solutions than other methods.

For further research, if the HPSO we proposed is implemented to other sequential ordering problems, there are two aspects for discussion: (1) Modify particle position representation for better suitability to the problem. In the original PSO design, the particles search solutions in a continuous solution space. Although most sequential ordering problems can be represented by the priority-based representation, it may not suit the sequential ordering problems. Preference list-based representation or other representations will better suit the algorithm for sequential ordering problems. (2) Design other particle movement methods and particle velocity for the modified particle position representation. In addition, which particle movement method or particle velocity is better could be a further research topic. Table 5.6 shows the summary of the HPSO for JSSP.

Table 5.6 Summary of the HPSO for JSSP

		Components	The concept of this components
1	Particle Position Representation	Preference-list	The preference-list representation has more Lamarckian than priority-based representation
2	Particle Velocity	Tabu list	The swap operator is appropriate to preference-list representation. It also fits in with the concept of correlation and connectivity.
	Particle Movement	Swap operator	
3	Decoding Operator	G&T algorithm	The G&T algorithm can restrict the search area but not exclude the optimal solution.
4	Other Strategies	Diversification Local search	The diversification strategy can prevent particles rapped in local optima. The local search can further improve the solution quality,

5.8 Appendix

<u>The pseudo code of the PSO for JSSP is given below:</u>

initialize a population of particles with random positions.

for each particle *k* ***do***

  apply G&T algorithm to decode X^k into a schedule S^k.

  set the *k*th *pbest* solution ($pbest^k$) equal to S^k, $pbest^k \leftarrow S^k$.

end for

set *gbest* solution equal to the best $pbest^k$.

repeat

  update velocities according to Figure 5.3.

  for each particle *k* ***do***

    move particle *k* according to Figure 5.5.

    apply G&T algorithm to decode x^k into S^k.

    update *pbest* solutions and *gbest* solution according to Figure 5.6.

    apply tabu search on S^{k^*}.

    update *pbest* solutions and *gbest* solution according to Figure 5.6.

  end for

until maximum iterations is attained

CHAPTER 6

A PARTICLE SWARM OPTIMIZATION FOR THE OPEN SHOP SCHEDULING PROBLEM

This chapter presents a new particle swarm optimization (PSO) for the open shop scheduling problem (OSSP). Compared with the original PSO, we modified the particle position representation using priorities, and the particle movement using an insert operator. We also implemented a modified parameterized active schedule generation algorithm (mP-ASG) to decode a particle position into a schedule. In mP-ASG, we can reduce or increase the search area between non-delay schedules and active schedules by controlling the maximum delay time allowed. Furthermore, we hybridized our PSO with beam search. The computational results show that our PSO found many new best solutions of the unsolved problems.

6.1 Particle Position Representation

In the previous researches (Liaw, 2000) and (Prins, 2000), an OSSP solution is represented by a permutation list, which is an ordered list of operations. For example, the following is a permutation list for a 3-job 2-machine problem:

$$\begin{array}{lcccccc} index: & 1 & 2 & 3 & 4 & 5 & 6 \\ permutation: & [o_{11} & o_{22} & o_{21} & o_{23} & o_{13} & o_{12}] \end{array}$$

Where o_{ij} is the operation of job j that needs to be processed on machine *i*. In the list, the prior operation has higher priority to be put into schedule. In our algorithm, we represent the particle *k* position by an $m \times n$ priority matrix for an *n*-job *m*-machine problem, i.e.

$$X^k = \begin{bmatrix} x_{11}^k & x_{12}^k & \cdots & x_{1n}^k \\ x_{21}^k & x_{22}^k & \cdots & x_{2n}^k \\ & \vdots & & \\ x_{m1}^k & x_{m2}^k & \cdots & x_{mn}^k \end{bmatrix}.$$

Where x_{ij}^k denotes the priority of operation o_{ij} of particle k. We can transfer the permutation list to a priority matrix. We randomly set x_{ij}^k between $(p-0.5)$ and $(p+0.5)$, that is, $x_{ij}^k \leftarrow p + rand_2 - 0.5$, where p is the location of o_{ij} in the permutation list, and $rand_2$ is a random variable between 0 and 1. Therefore, the operation with smaller x_{ij}^k has higher priority to be put into schedule. The permutation list mentioned above can be randomly transferred to

$$X^k = \begin{bmatrix} 0.6 & 5.8 & 5.2 \\ 3.3 & 2.1 & 3.9 \end{bmatrix}.$$

When we implement insert operator, the advantage of the priority matrix is to reduce the computation complexity. For example, in the permutation list mentioned above, when we want to insert o_{12} into the third location of the permutation list, we first need to move o_{13} to the sixth location, move o_{23} to fifth location, move o_{21} to the fourth location, and than insert o_{12} to the third location, i.e.

$$\begin{array}{lcccccc} index: & 1 & 2 & 3 & 4 & 5 & 6 \\ permutation: & [o_{11} & o_{22} & o_{12} & o_{21} & o_{23} & o_{13}] \end{array}$$

On the average, the insert operator costs $O(n/2)$ steps for each insertion.

However, if we implement insert operator on a priority matrix, we only need to set $x_{ij}^k \leftarrow 3 + rand_2 - 0.5$ when we want to insert o_{12} to the third location of the permutation list. Therefore, there is only one step needed for each insertion. For example, if the random number $rand_2$ equals 0.9, the following is the X^k after o_{12} inserted into the third location:

$$X^k = \begin{bmatrix} 0.6 & 3.4 & 5.2 \\ 3.3 & 2.1 & 3.9 \end{bmatrix}.$$

In X^k, both o_{12} and o_{21} are on the third location of the permutation list (i.e. both x_{12}^k and x_{21}^k are between 2.5 and 3.5), but the values of x_{12}^k and x_{21}^k are different due to the random variable $rand_2$. Consequently, the order of o_{12} and o_{21} to be put into the schedule is also randomly determined. This means that, if the priority matrix shows that there are more than two operations located on the same location of the permutation list, the order of the operations on the same location is randomly determined by the random variable $rand_2$. The detail of the particle movement operator is described in section 6.3.

The PSO we proposed differs from the original PSO design in that the *gbest* and *pbest* solutions do not record the best positions found so far, but rather the best schedules generated by the decoding operator. In our algorithm, the decoding operator puts the operations one by one into a schedule. The sequential order of the operations that the decoding operator puts them into the schedule can be formulated as an operation sequence. Then we can transfer the operation sequence to an $m \times n$ matrix S^k, i.e.

$$S^k = \begin{bmatrix} s_{11}^k & s_{12}^k & \cdots & s_{1n}^k \\ s_{21}^k & s_{22}^k & \cdots & s_{2n}^k \\ & \vdots & & \\ s_{m1}^k & s_{m2}^k & \cdots & s_{mn}^k \end{bmatrix}.$$

Where s_{ij}^k denotes the location of operation o_{ij} in the operation sequence. For example, if the operation sequence generated by the decoding operator is as follows:

$$
\begin{array}{llllllll}
\textit{index}: & & 1 & 2 & 3 & 4 & 5 & 6 \\
\textit{operation sequence}: & [& o_{13} & o_{22} & o_{12} & o_{23} & o_{11} & o_{21}]
\end{array}
$$

we can transfer the operation sequence to the matrix S^k:

$$S^k = \begin{bmatrix} 5 & 3 & 1 \\ 6 & 2 & 4 \end{bmatrix}.$$

When we update *pbest* and *gbest* solutions, we do not record the best positions found so far (i.e. X^k), but rather the locations of the operations in the operation sequence, which is generated by the decoding operator (i.e. S^k).

6.2 Particle Velocity

In the original PSO, the update of the particle velocity depends on the g*best* and the *pbest* solutions; and then each particle moves according to its velocity. There are two purposes of the particle velocity: (a) moving a particle toward the *gbest* and the *pbest* solutions; (b) keeping inertia to prevent particles getting trapped in local optima.

In our PSO, the particle velocity does not move particles toward the *pbest* and the *gbest* solution, but only prevents particles getting trapped in local optima. If the priority value is increased or decreased in this iteration, we keep the priority value increasing or decreasing at the beginning of the next iteration with probability w, which is the inertia weight in our PSO. The larger the w, the more iterations the priority value keeps increasing or decreasing, and it is harder for the particle to return to the current position. For an *n*-job *m*-machine problem, we represent the velocity of particle *k* by an *m*×*n* matrix, i.e.

$$V^k = \begin{bmatrix} v_{11}^k & v_{12}^k & \cdots & v_{1n}^k \\ v_{21}^k & v_{22}^k & \cdots & v_{2n}^k \\ & \vdots & & \\ v_{m1}^k & v_{m2}^k & \cdots & v_{mn}^k \end{bmatrix}.$$

Where v_{ij}^k is the velocity of the operation o_{ij} of particle k, $v_{ij}^k \in \{-1, 0, 1\}$.

Compared with the original PSO, the velocity in our PSO only considers whether the value of x_{ij}^k is larger or smaller than $pbest_{ij}^k$ ($gbest_{ij}$), but does not consider the distance from x_{ij}^k to $pbest_{ij}^k$ ($gbest_{ij}$). In our algorithm, if x_{ij}^k is reduced in this iteration, that is, $pbest_{ij}^k$ ($gbest_{ij}$) is smaller than x_{ij}^k and x_{ij}^k is set toward $pbest_{ij}^k$ ($gbest_{ij}$), we set $v_{ij}^k \leftarrow -1$, indicating that x_{ij}^k should be kept decreasing one (i.e. $x_{ij}^k \leftarrow x_{ij}^k - 1$) in the next iteration with probability w. On the contrary, if x_{ij}^k is increased in this iteration, that is, $pbest_{ij}^k$ ($gbest_{ij}$) is larger than x_{ij}^k and x_{ij}^k is set toward $pbest_{ij}^k$ ($gbest_{ij}$), we set $v_{ij}^k \leftarrow 1$, indicating that x_{ij}^k should be keep increasing one (i.e. $x_{ij}^k \leftarrow x_{ij}^k + 1$) in the next iteration with probability w.

The velocity can be controlled by inertia weight w. We randomly update velocities at the beginning of the iteration. For each particle k and operation o_{ij}, if v_{ij}^k does not equal 0, v_{ij}^k will be set to 0 with the probability ($1-w$). This means that if x_{ij}^k keeps increasing or decreasing, x_{ij}^k stops keeping increasing or decreasing in this iteration with the probability ($1-w$). The pseudo code for updating velocities is given in Figure 6.1. The complexity of updating particle velocity for each particle is $O(mn)$, where mn is the number of operations.

```
for each particle k and operation o_ij do
    rand ~ U(0,1)
    if (v_ij^k ≠ 0) and (rand ≥ w) then
        v_ij^k ← 0
    end if
end for
```

Figure 6.1 The pseudo code of updating velocities.

6.3 Particle Movement

In our PSO, the particle movement is based on the insert operator. As mentioned in section 6.1, we set $x_{ij}^k \leftarrow p + rand_2 - 0.5$ if we want to insert o_{ij} to the p^{th} location in the permutation list. In addition, the location of operation o_{ij} in the operation sequence of k^{th} *pbest* and *gbest* solution are $pbest_{ij}^k$ and $gbest_{ij}$ respectively. When particle k moves, for all o_{ij}, if v_{ij}^k equals 0, the x_{ij}^k will be set to $pbest_{ij}^k + rand_2 - 0.5$ with probability c_1 and set to $gbest_{ij} + rand_2 - 0.5$ with probability c_2, where $rand_2$ is a random variable between 0 and 1, and c_1 and c_2 are constants between 0 and 1, and $c_1 + c_2 \le 1$. For example, assume that V^k, X^k, $pbest^k$, $gbest$, c_1, and c_2 are as follows:

$$V^k = \begin{bmatrix} -1 & 0 \\ 0 & 0 \end{bmatrix}, \quad X^k = \begin{bmatrix} 2.5 & 3.3 \\ 1.3 & 4.2 \end{bmatrix}, \quad pbest^k = \begin{bmatrix} 1 & 4 \\ 3 & 2 \end{bmatrix}, \quad gbest = \begin{bmatrix} 3 & 4 \\ 1 & 2 \end{bmatrix},$$

$$c_1 = 0.7, \quad c_2 = 0.1.$$

For o_{11}:

Because $v_{11}^k \neq 0$, $x_{11}^k \leftarrow x_{11}^k + v_{11}^k$, that is, $x_{11}^k = 1.5$.

For o_{12}:

Because $v_{12}^k = 0$, randomly generated random variable $rand_1$ = 0.6.

Because $rand_1 \le c_1$, randomly generated random variable $rand_2$ = 0.3.

Because $pbest_{12}^k \ge x_{12}^k$, set $v_{12}^k \leftarrow 1$, and then $x_{12}^k \leftarrow pbest_{12}^k + rand_2$ - 0.5, that is, $x_{12}^k = 3.8$.

For o_{21}:

Because $v_{21}^k = 0$, generate random variable $rand_1$ = 0.9.

Because $rand_1 > c_1 + c_2$, we do not change the value of x_{21}^k.

For o_{22}:

Because $v_{22}^k = 0$, randomly generated random variable $rand_1$ = 0.75.

Because $c_1 < rand_1 \le c_1 + c_2$, randomly generated random variable $rand_2$ = 0.8.

Because $gbest_{22} < x_{22}^k$, set $v_{22}^k \leftarrow$ -1, and then $x_{22}^k \leftarrow gbest_{22} + rand_2$ - 0.5, that is, $x_{22}^k = 2.3$.

Therefore, the V^k and X^k after particle k is moved are:

$$V^k = \begin{bmatrix} -1 & 1 \\ 0 & -1 \end{bmatrix} \text{ and } X^k = \begin{bmatrix} 1.5 & 3.8 \\ 1.3 & 2.3 \end{bmatrix}.$$

Furthermore, we adopt a mutation operator in our algorithm. After a particle moves to a new position, we randomly choose an operation and then mutate its priority value x_{ij}^k disregarding v_{ij}^k. If $x_{ij}^k \le (mn/2)$, we randomly set the value of x_{ij}^k between $(mn-n)$ and mn, and set $v_{ij}^k \leftarrow 1$. If $x_{ij}^k > (mn/2)$, we randomly set the value of x_{ij}^k between 0 and n, and set $v_{ij}^k \leftarrow -1$. The pseudo code of particle movement is given in Figure 6.2. The complexity of each particle movement is $O(mn)$.

for $i \leftarrow 1$ **to** m **do** //for all operations

for $j \leftarrow 1$ **to** n **do**

if $(v_{ij}^k = 0)$ **then**

$rand_1 \sim U(0,1)$

if $(rand_1 \leq c_1)$ **then**

if $(pbest_{ij}^k \geq x_{ij}^k)$ **then** $v_{ij}^k = 1$ **else** $v_{ij}^k = -1$

$rand_2 \sim U(0,1)$

$x_{ij}^k \leftarrow pbest_{ij}^k + rand_2 - 0.5$

end if

if $(c_1 < rand_1 \leq c_1 + c_2)$ **then**

if $(gbest_{ij} \geq x_{ij}^k)$ **then** $v_{ij}^k = 1$ **else** $v_{ij}^k = -1$

$rand_2 \sim U(0,1)$

$x_{ij}^k \leftarrow gbest_{ij} + rand_2 - 0.5$

end if

end if

end for

end for

// ↓ **mutation operator**

$(i, j) \leftarrow$ Randomly choose an operation

$rand_1 \sim U(0,1)$

if $(x_{ij}^k \leq (m \times n / 2))$ **then**

$x_{ij}^k \leftarrow m \times n - n \times rand_1$; $v_{ij}^k = 1$

else

$x_{ij}^k \leftarrow n \times rand_1$; $v_{ij}^k = -1$

end if

// ↑ **mutation operator**

Figure 6.2 The pseudo code of particle movement.

6.4 Decoding Operators

If we want to decode a particle position matrix into a schedule, we can put the operations into a schedule by ascending order of x_{ij}^k. However, if we directly put the operations into a schedule by ascending order of x_{ij}^k, it will be a semi-active schedule. The set of semi-active schedules is very large and has poor quality in terms of makespan. Therefore, we need an efficient decoding operator to decode the particle position into a schedule within a schedule set, which set size is smaller and with better solution quality.

Because there is no precedence relation between the operations of each job in OSSP, the solution space of OSSP is much larger than flow shop and job shop scheduling problems. Therefore, a decoding operator, which can reduce the search area but does not exclude the optimal solution, will improve the efficiency of PSO.

The optimal OSSP solution should be an active schedule. In an active schedule, the processing sequence is such that no operation can be started any earlier without delaying another operation (French , 1982). However, the set of active schedules is usually very large and with poor quality in terms of makespan. The set of non-delay schedules is a subset of active schedules. In a non-delay schedule, no machine is kept idle at a time when it could begin processing other operations (French , 1982). The set of non-delay schedules is much smaller than the active schedules, and usually with better quality in terms of makespan. However, the optimal solution is not a non-delay schedule in most of the test problems.

We will describe four different decoding operators that generate schedules between a non-delay set and an active set. All the decoding operators are based on Giffler and Thompson's algorithm (Giffler & Thompson, 1960), which can decode a particle position into an active schedule. We briefly describe the G&T algorithm for OSSP in Figure 6.3. If we modify the operation set O in the G&T algorithm as follows, this algorithm will decode the particle position into a non-delay schedule:

Determine $s^* \leftarrow \min_{o_{ij} \in \Omega} \{s_{ij}\}$.

Identify the operation set $O \leftarrow \{o_{ij} \mid s_{ij} = s^*, o_{ij} \in \Omega\}$.

Moreover, we can generate schedules between a non-delay set and an active set by modifying the operation set O in the G&T algorithm.

Notation:

o_{ij} : the operation of job j that needs to be processed on machine i.

S : the partial schedule that contains scheduled operations.

Ω : the set of schedulable operations.

s_{ij} : the earliest time at which $o_{ij} \in \Omega$ could be started.

p_{ij} : the processing time of o_{ij}.

f_{ij} : the earliest time at which $o_{ij} \in \Omega$ could be finished, $f_{ij} = s_{ij} + p_{ij}$.

G&T algorithm for OSSP:

Initialize $S \leftarrow \phi$; Ω is initialized to contain all operations.

repeat

Determine $f^* \leftarrow \min_{o_{ij} \in \Omega}\{f_{ij}\}$.

Identify the operation set $O \leftarrow \{o_{ij} \mid s_{ij} < f^*, o_{ij} \in \Omega\}$.

Choose o_{ij}^* from the operation set O with the smallest x_{ij}^k, add o_{ij}^* to S, and assign s_{ij} as the starting time of o_{ij}^*.

Delete o_{ij}^* from Ω.

Until Ω is empty.

Figure 6.3 The G&T algorithm for OSSP.

6.4.1 Decoding Operator 1 (A-ND)

The first decoding operator randomly identifies the operation set O in the active set or the non-delay set. It was first proposed in (Prins, 2000) and was also proposed in (Blum, 2005). It can be reached by modifying the operation set O in the G&T algorithm:

Determine $s^* \leftarrow \min_{o_{ij} \in \Omega}\{s_{ij}\}$ and $f^* \leftarrow \min_{o_{ij} \in \Omega}\{f_{ij}\}$.

Randomly identify the operation set O equal to $\{o_{ij} \mid s_{ij} < f^*, o_{ij} \in \Omega\}$

or $\{o_{ij} \mid s_{ij} = s^*, o_{ij} \in \Omega\}$ with probability 0.5 respectively.

To determine s^* and f^* costs $O(mn)$ each, and to choose o_{ij}^* from the operation set O with the smallest x_{ij}^k also costs $O(mn)$. There are mn operations needing to be schedule, hence the complexity of *decoding operator 1* is $O(m^2n^2)$.

6.4.2 Decoding Operator 2 (mP-ASG)

Gonçalves et al. (2005) proposed a parameterized active schedule generation algorithm (P-ASG) for the job shop scheduling problem. We (Sha & Hsu, 2006) modified the algorithm based on the G&T algorithm for better performance and so it is easier to implement. The modified parameterized active schedule generation algorithm (mP-ASG) can also be applied to OSSP.

The basic idea of parameterized active schedules is to control the search area by controlling the delay times that each operation is allowed. If all of the delay times are equal to zero, the set of parameterized active schedules is equivalent to non-delay schedules. On the contrary, if all of the delay times are equal to infinity, the set of parameterized active schedules is equivalent to the active schedules. Therefore, the set of parameterized active schedules is between the non-delay schedules and the active schedules. Fig. 5 (Gonçalves et al., 2005) shows the relationship between the sets of semi-active, active, non-delay, and parameterized active schedules. It also shows that the size of the parameterized active schedules depends upon the delay times that each operation is allowed.

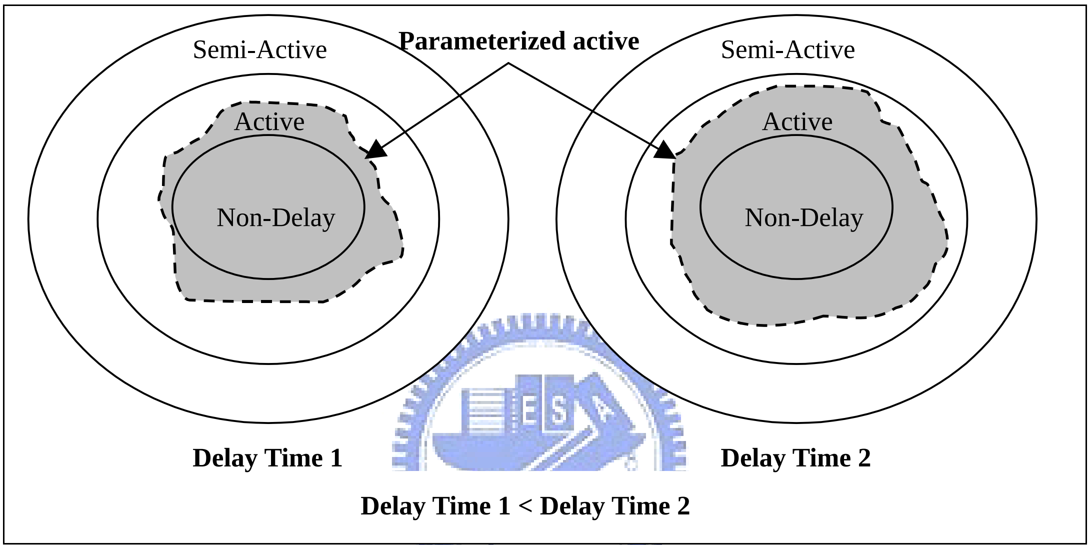


Figure 6.4 Parameterized active schedules.

In our mP-ASG, we defined the delay time $delay \leftarrow (f^* - s^*) \times delayweight$, so that modifying the operation set O in the G&T algorithm can generate the parameterized active schedules:

Determine $s^* \leftarrow \min_{o_{ij} \in \Omega}\{s_{ij}\}$, $f^* \leftarrow \min_{o_{ij} \in \Omega}\{f_{ij}\}$, and $delay \leftarrow (f^* - s^*) \times$ $delayweight$.

Identify the operation set $O \leftarrow \{o_{ij} \mid s_{ij} \le s^* + delay, o_{ij} \in \Omega\}$.

The value of $delayweight$ is between 0 and 1 for controlling the search area. When $delayweight$ equals 0, the operation set O is equal to the non-delay set. On the contrary, when *delayweight* equals 1, the operation set O is equal to the active set. In our PSO, *delayweight* increases linearly from 0 to 1 during a run. This means that the PSO searches solutions near to the non-delay set in earlier iterations for better initial solutions, and then enlarges the search area to prevent missing the optimal solution. As with *decoding operator 1*, the complexity of *decoding operator 2* is $O(m^2 n^2)$.

6.4.3 Decoding Operator 3 (mP-ASG2)

We implement fathoming constraints in mP-ASG to further reduce the size of the operation set O. The fathoming constraints originate from one of the fathoming criteria in the branch-and-bound algorithm. One of the fathoming criteria in the branch-and-bound algorithm is that, if the lower bound of a partial solution is larger than the current upper bound (for a minimization problem), the partial solution is fathomed. Hence the partial solution need not be investigated any further because it does not yield a better solution than the current upper bound.

In OSSP, the lower bound of a partial schedule after o_{ij} is put into it can be roughly estimated by $\max\{s_{ij} + r_i^M, s_{ij} + r_j^J\}$, where r_i^M (r_j^J) is the sum of the remaining process times on machine i (job j). Therefore, we can fathom whether the lower bound of a partial schedule after o_{ij} is put into it is larger than current upper bound — the makespan of the *gbest* solution, $C_{\max}(gbest)$, by whether o_{ij} fits in with the following constraint:

$$\max\{s_{ij} + r_i^M, s_{ij} + r_j^J\} \le C_{\max}(gbest) \tag{5.1}$$

that is,

$$s_{ij} \le C_{\max}(gbest) - r_i^M \tag{5.2}$$

and

$$s_{ij} \le C_{\max}(gbest) - r_j^J \tag{5.3}$$

However, we cannot ensure that the lower bounds of all the generated partial schedules are less than $C_{\max}(gbest)$. If the lower bound of a partial schedule is larger than $C_{\max}(gbest)$, there is no unscheduled operation to fit in with constraints (5.2) and (5.3). To avoid such a situation, we modify the constraints by:

$$s_{ij} \le \max\{s_i^M, C_{\max}(gbest) - r_i^M\} \tag{5.4}$$

and

$$s_{ij} \le \max\{s_j^J, C_{\max}(gbest) - r_j^J\} \tag{5.5}$$

where $s_i^M = \min_{\forall j}\{s_{ij} \mid o_{ij} \in \Omega\}$ and $s_j^J = \min_{\forall i}\{s_{ij} \mid o_{ij} \in \Omega\}$. This means that when there is no unscheduled operation to fit in with constraints (5.2) and (5.3) on some machines or jobs in the partial schedule, we prevent the machines or jobs being idle as much as possible.

We name the constraints (5.4) and (5.5) "*fathoming constraints*". The fathoming constraints are used to reduce further the size of the operation set O. The parameterized active schedules with fathoming constraints can be generated by modifying the operation set O in the G&T algorithm:

Determine $s^* \leftarrow \min_{o_{ij}\in\Omega}\{s_{ij}\}$, $f^* \leftarrow \min_{o_{ij}\in\Omega}\{f_{ij}\}$, $delay \leftarrow (f^* - s^*) \times delayweight$, and s_i^M, s_j^J, r_i^M, r_j^J for all i, j.

Identify the operation set $O \leftarrow \{o_{ij} \mid s_{ij} \le s^* + delay, o_{ij} \in \Omega,$

$s_{ij} \le \max\{s_i^M, C_{\max}(gbest) - r_i^M\}, s_{ij} \le \max\{s_j^J, C_{\max}(gbest) - r_j^J\}\}$.

To determine s^*, f^* costs $O(mn)$, and to determine all the s_i^M, s_j^J, r_i^M, r_j^J also costs $O(mn)$, so the complexity of *decoding operator 3* is $O(m^2n^2)$.

6.4.4 Decoding Operator 4 (mP-ASG2+BS)

Blum (2005) hybridizes the solution construction mechanism of ant colony optimization (ACO) with beam search (BS). He also shows that the ACO hybridized with BS performs much better than standard ACO. Since both the solution construction mechanism of ACO and our PSO are based on the G&T algorithm, we can also hybridize the decoding operators with BS in the same way.

BS is an adaptation of the branch-and-bound method in which only some nodes are evaluated in the search tree. In each step, k_{bw} partial solutions at most are allowed to extend, and the partial solutions can only extend in k_{ext} possible ways at most. The extendable partial solution set, B, is called the *beam* and the largest size of the set, k_{bw}, is called the *beam width*. The partial schedule is evaluated by its lower bound, which can be easily computed as follows (Gonzalez & Sahni, 1976):

$$LB = \max\{X, Y\} \tag{5.6}$$

where

$$X = \max\{s_j^J + r_j^J \mid \forall o_{ij} \in \Omega\} \tag{5.7}$$

$$Y = \max\{s_i^M + r_i^M \mid \forall o_{ij} \in \Omega\} \tag{5.8}$$

We chose the decoding operator mP-ASG2 to hybridize with BS, because it outperforms others as we will illustrate in section 6.6.1. We briefly describe the procedure of mP-ASG2+BS in Figure 6.5.

Initialize $S \leftarrow \phi$; choose o_{ij}^* with the smallest x_{ij}^k; add o_{ij}^* to S; $B \leftarrow \{S\}$.

repeat

Initialize $B_{ext} \leftarrow \phi$.

for each partial schedule S in B ***do***

$O_{ext} \leftarrow$Select at most k_{ext} operations from the operation set O identified as mP-ASG2 with the smallest x_{ij}^k.

for each $o_{ij} \in O_{ext}$ ***do***

$S' \leftarrow$Extend S by adding o_{ij}, and then add S' to B_{ext}.

end for

end for

if $|B_{ext}| \leq k_{ext}$ ***then***

$B \leftarrow B_{ext}$.

else

$B \leftarrow$ Select k_{ext} best partial schedules in B_{ext} with the smallest lower bound.

end if

until each S in B contains all operations.

Output the best schedule with the smallest makespan in B as the schedule generated by particle k.

Figure 6.5 The pseudo code of the decoding operator mP-ASG2+BS.

In *decoding operator 4*, for each partial schedule S in B, it costs $O(mn)$ to select at most k_{ext} operations from the operation set O identified as mP-ASG2 with the smallest x_{ij}^{k}. There are mn levels of the search tree and k_{bw} partial schedules in B at most. Therefore, the complexity of *decoding operator 4* is $O(k_{ext}m^2n^2)$. Blum (2005) sets $k_{bw} = mn$, so the complexity of Beam-ACO to construct a solution is $O(m^3n^3)$. It is evident that the computation time of BS massively increases following the problem size increasing. We modified the parameter settings in (Blum, 2005) to prevent such a situation as much as possible. In our algorithm, There is only one node in the first level of the search tree, and we set $k_{ext} = 2$, the smallest setting value of k_{ext}, and $k_{bw} = 2n$. The parameter k_{bw} is dependent upon the number of jobs (n) rather than the number of operations (mn), so the computation time of BS is more insensitive to the problem size than (Blum, 2005). Consequently, the complexity of *decoding operator 4* is $O(m^2n^3)$. We will test and compare these four decoding operators in section 6.6.1.

6.5 The Diversification Strategy

We also implement the diversification strategy as described in section 5.4. The diversification strategy keeps the *pbest* solutions different (i.e. it keeps the makespans of *pbest* solutions different). In the diversification strategy, the *pbest* solution of each particle is not the best solution found by the particle itself, but one of the best *N* solutions found by the swarm so far where *N* is the size of the swarm. The complexity of updating the g*best* and *pbest* solutions with diversification strategy for each particle is $O(Nmn)$.

6.6 Computational Results

We tested our PSO on the benchmark problems proposed by Taillard (1993) (tai_*×*_*), Brucker et al. (1997) (j*-per*-*), and Guéret and Prins (1999) (gp*-*).

The program was coded in Visual C++ and run 20 times on each problem under Windows XP on an AMD Athlon 1800+ PC. In the preliminary experiment, four swarm sizes *N* (10, 30, 60, 100) were tested, where *N*=60 was superior and used for all further studies. The parameters c_1 and c_2 were tested between 0.1 and 0.7 in increments of 0.2. The inertia weight w was decreased linearly from $w_{\max}$ to $w_{\min}$ during a run. The parameter $w_{\max}$ was tested on 0.5, 0.7, and 0.9. Another parameter $w_{\min}$ was tested on 0.1, 0.3, and 0.5. The settings $c_1 = 0.7$, $c_2 = 0.1$, $w_{\max} = 0.9$, and $w_{\min} = 0.3$ were superior. As mentioned before, the parameter *delayweight* in mP-ASG, mP-ASG2, and mP-ASG2+BS increases linearly from 0 to 1 during a run. All of the numeric parameters are determined empirically. The program only stops at the CPU time limits even if the lower bound reached. We set the CPU time limit of each run referring to ACO (Blum, 2005). Because the results of ACO (Blum, 2005) were obtained on an 1100 MHz PC, and the machine is slower than ours, we set the time limits of each run at only half of ACO (Blum, 2005) to confirm that the spent CPU time is not longer than ACO (Blum, 2005).

6.6.1 Comparison of Decoding Operators

We tested these four decoding operators on the hardest problem sets: j8-per*-* and gp10*-*. We set the time limits of each run on the j8-per*-* and gp10-* to 245 and 500 CPU seconds respectively. As mentioned above, the time limit of each run is half of ACO (Blum, 2005). Each program was run 20 times on each problem. Moreover, to show whether the mutation operator can improve the solution quality, we also tested our PSO, which does not implement the mutation operator.

Table 6.1(a) shows the computational results of the four decoding operators, and Table 6.1(b) shows the computational results, in which we do not implement the mutation operator. In the tables, the terms ‘Best’ and ‘Average’ mean the best and average solution of the 20 runs respectively. The 'BKS' is the best-known solution. If the BKS is not proved to be optimal, it is bracketed. The BKSs in the tables include our method. If the BKS is marked an asterisk, it means that the value is obtained by one of our PSOs. If the solution value is indicated in bold, it means that the value is equal to the BKS. For each problem, the solution gap is defined by $(C_{\max} - BKS)/BKS$ as a percentage, where $C_{\max}$ is the makespan of the solution. Therefore, the average gap is obtained by averaging the solution gaps.

Compare the results between Table 6.1(a) and Table 6.1(b). All the average gaps in Table 6.1(a) are smaller than those in Table 6.1(b). This shows that the mutation operator improves the solution quality. In Table 6.1(a) and Table 6.1(b), the results show that the average gap of mP-ASG2 is less than A-ND and mP-ASG. It is evident that the fathoming constraints in mP-ASG2 can successfully reduce the search area without excluding the optimal solution, and thus obtain better results. In Table 6.1(a), the mP-ASG2+BS obtains the best result on j8-per*-* test problems, and the mP-ASG2 obtains the best result on gp10-* test problems. Therefore, we chose mP-ASG2 and mP-ASG2+BS for further tests, and compared the results with other metaheuristics.

Table 6.1(a) Computational results of four decoding operators with mutation operator

Problem	BKS	A-ND		mP-ASG		mP-ASG2		mP-ASG2+BS	
		Best	Average	Best	Average	Best	Average	Best	Average
j8-per0-1	(1039)	1059	1067.50	**1039**	1050.65	1045	1051.65	**1039**	1043.25
j8-per0-2	(1052)	1057	1066.60	1055	1063.45	**1052**	1058.30	**1052**	1053.60
j8-per10-0	(1020)	1029	1034.50	**1020**	1034.80	1024	1033.05	**1020**	1026.10
j8-per10-1	(1002)*	1011	1019.95	1008	1015.35	1008	1018.25	**1002**	1007.55
j8-per10-2	(1002)*	1009	1024.05	**1002**	1014.70	**1002**	1012.45	**1002**	1005.95
j8-per20-0	1000	1003	1006.55	**1000**	1000.95	**1000**	1000.90	**1000**	1000.60
j8-per20-1	1000	**1000**	**1000.00**	**1000**	**1000.00**	**1000**	**1000.00**	**1000**	**1000.00**
j8-per20-2	1000	1001	1004.70	**1000**	1004.10	**1000**	1003.15	**1000**	**1000.00**
Average gap		0.660%	1.334%	0.111%	0.846%	0.196%	0.771%	0.000%	0.271%
gp10-01	(1093)*	**1093**	1096.00	**1093**	1096.25	**1093**	1097.40	**1093**	1096.75
gp10-02	(1097)*	**1097**	1097.15	**1097**	1097.05	**1097**	**1097.00**	**1097**	1099.05
gp10-03	(1081)	1087	1094.85	**1081**	1087.90	**1081**	1087.10	1084	1090.30
gp10-04	(1083)*	1089	1091.30	1086	1089.10	1086	1089.10	**1083**	1092.10
gp10-05	(1073)*	1084	1091.15	1082	1087.70	**1073**	1086.50	1082	1092.15
gp10-06	(1071)*	**1071**	1092.80	**1071**	**1071.00**	**1071**	**1071.00**	**1071**	1074.25
gp10-07	(1080)*	1081	1081.40	1081	1081.60	**1080**	1080.95	1081	1081.05
gp10-08	(1095)*	**1095**	1096.45	**1095**	1097.80	**1095**	1097.25	1097	1097.55
gp10-09	(1115)*	1116	1122.15	1116	1118.30	**1115**	1117.80	1123	1127.00
gp10-10	(1092)	**1092**	1094.95	**1092**	1092.05	**1092**	1092.15	**1092**	1093.95
Average gap		0.232%	0.724%	0.130%	0.358%	0.028%	0.335%	0.211%	0.590%

The time limits for each run on problems j8-per*-* and gp10-* are 245 and 500 CPU seconds respectively.
*The BKS is found by one of our PSOs.

Table 6.1 (b) Computational results of four decoding operators without mutation operator

Problem	BKS	A-ND		mP-ASG		mP-ASG2		mP-ASG2+BS	
		Best	Average	Best	Average	Best	Average	Best	Average
j8-per0-1	(1039)	1056	1064.85	1046	1054.95	1047	1054.80	1042	1043.95
j8-per0-2	(1052)	1057	1068.25	1064	1064.00	1064	1064.00	1053	1058.85
j8-per10-0	(1020)	1030	1039.65	1022	1035.25	1022	1035.80	**1020**	1028.30
j8-per10-1	(1002)*	1016	1023.25	1009	1015.70	1009	1015.65	**1002**	1008.20
j8-per10-2	(1002)*	1009	1024.20	**1002**	1015.50	**1002**	1014.30	**1002**	1006.40
j8-per20-0	1000	1002	1008.30	**1000**	1000.80	**1000**	1001.05	**1000**	1000.45
j8-per20-1	1000	**1000**	1000.05	**1000**	**1000.00**	**1000**	**1000.00**	**1000**	**1000.00**
j8-per20-2	1000	**1000**	1006.60	**1000**	1005.10	**1000**	1005.00	**1000**	**1000.00**
Average gap		0.673%	1.474%	0.339%	0.934%	0.351%	0.926%	0.048%	0.381%
gp10-01	(1093)*	1095	1100.00	1095	1101.05	1095	1099.45	**1093**	1097.40
gp10-02	(1097)*	1099	1109.25	**1097**	1100.55	**1097**	1100.75	1098	1100.35
gp10-03	(1081)	1088	1100.90	1088	1096.05	1087	1095.00	1085	1093.00
gp10-04	(1083)*	1090	1094.50	1090	1090.90	1090	1091.75	1087	1092.70
gp10-05	(1073)*	1086	1094.90	1084	1092.25	1083	1092.20	1082	1090.10
gp10-06	(1071)*	**1071**	1093.00	**1071**	1078.80	**1071**	1077.35	**1071**	1072.05
gp10-07	(1080)*	1082	1087.60	1081	1081.65	1081	1081.80	1081	1081.15
gp10-08	(1095)*	1098	1099.80	1097	1100.45	1098	1099.65	1096	1098.60
gp10-09	(1115)*	1117	1127.40	1116	1131.55	1116	1126.40	1125	1131.35
gp10-10	(1092)	**1092**	1097.40	**1092**	1094.00	**1092**	1094.55	**1092**	1093.85
Average gap		0.393%	1.218%	0.335%	0.870%	0.324%	0.792%	0.332%	0.721%

The time limits for each run on problems j8-per*-* and gp10-* are 245 and 500 CPU seconds respectively.
*The BKS is found by one of our PSOs.

6.6.2 Comparison with Other Metaheuristics

We compared our PSOs with the GA proposed by Liaw (2000) (denoted by GA-Liaw), the GA proposed by Prins (2000) (denoted by GA-Prins), and the ACO hybridized with beam search proposed by Blum (2005) (denoted by Beam-ACO). Our results were obtained by 20 runs on each problem, and the time limit of each run is half of Beam-ACO. The program only stops at the CPU time limits even if the lower bound reached. It is important to remember that both our PSO and Beam-ACO performed 20 runs, but the two GAs, GA-liaw and GA-Prins, performed only one run. Moreover, in our PSO and Beam-ACO, the computation time of each run is substantially longer than GA-liaw and GA-Prins. Therefore, we will mainly compare our computational results with Beam-ACO.

Table 6.2 shows the results of the test problems proposed by Taillard (1993). The term '*t*' is the average time needed by one run to get its best makespan value. Compared with other test problem sets, Taillard test problems are easier to solve. Therefore, Beam-ACO and PSO-mP-ASG2+BS obtained all of the optimal solutions, but PSO-mP-ASG2+BS is slightly better than Beam-ACO on average gaps. The *t* of PSO-mP-ASG2 and PSO-mP-ASG2+BS are quite similar on the problem set tai_4×4_*. The reason for this is that the search area of these two PSOs is restricted by the parameter *delayweight* . In the decoding operator, the *delayweight* increases linearly form 0 to 1 during a run. This means that the search area of PSO is near to the non-delay set in earlier iterations, and then enlarges after *delayweight* increases. If the problem size is quite small but the *t* is rather large, it means that the optimal solution is far from the non-delay set, and the PSO can only obtain the optimal solution when *delayweight* increases.

Table 6.3 shows the results of the test problems proposed by Brucker et al.

(1997). Although there are two best solutions obtained by PSO-mP-ASG2+BS worse than Beam-ACO on test problem j7-per0-0 and j7-per20-1, all the average gaps obtained by PSO-mP-ASG2+BS are better than Beam-ACO. Furthermore, PSO-mP-ASG2+BS obtained new best-known solutions on test problem j8-per10-1 and j8-per10-2.

Table 6.2 Results of the test problems proposed by Taillard (1993)

Problem	BKS	GA-Liaw	GA-Prins	Beam-ACO		PSO-mP-ASG2			PSO-mP-ASG2+BS			Time
				Best	Average	Best	Average	t	Best	Average	t	limit(s)
tai_4×4_1	193	**193**	**193**	**193**	**193.0**	**193**	**193.0**	4.7	**193**	**193.0**	4.7	8
tai_4×4_2	236	**236**	239	**236**	**236.0**	**236**	**236.0**	3.7	**236**	**236.0**	3.7	8
tai_4×4_3	271	**271**	**271**	**271**	**271.0**	**271**	**271.0**	4.7	**271**	**271.0**	5.0	8
tai_4×4_4	250	**250**	**250**	**250**	**250.0**	**250**	**250.0**	2.3	**250**	**250.0**	2.3	8
tai_4×4_5	295	**295**	**295**	**295**	**295.0**	**295**	**295.0**	2.9	**295**	**295.0**	2.9	8
tai_4×4_6	189	**189**	**189**	**189**	**189.0**	**189**	**189.0**	2.1	**189**	**189.0**	2.1	8
tai_4×4_7	201	**201**	**201**	**201**	**201.0**	**201**	**201.0**	6.5	**201**	**201.0**	6.5	8
tai_4×4_8	217	**217**	**217**	**217**	**217.0**	**217**	**217.0**	6.4	**217**	**217.0**	7.0	8
tai_4×4_9	261	**261**	**261**	**261**	**261.0**	**261**	**261.0**	1.3	**261**	**261.0**	1.3	8
tai_4×4_10	217	**217**	221	**217**	**217.0**	**217**	**217.0**	1.7	**217**	**217.0**	1.7	8
Average gap		0.000%	0.311%	0.000%	0.000%	0.000%	0.000%		0.000%	0.000%		
tai_5×5_1	300	**300**	301	**300**	**300.0**	**300**	**300.0**	1.2	**300**	**300.0**	1.1	25
tai_5×5_2	262	**262**	263	**262**	**262.0**	**262**	**262.0**	2.1	**262**	**262.0**	2.1	25
tai_5×5_3	323	**323**	335	**323**	**323.0**	**323**	**323.0**	14.4	**323**	**323.0**	14.4	25
tai_5×5_4	310	**310**	316	**310**	**310.0**	**310**	**310.0**	10.4	**310**	**310.0**	10.1	25
tai_5×5_5	326	**326**	330	**326**	**326.0**	**326**	**326.0**	9.9	**326**	**326.0**	9.3	25
tai_5×5_6	312	**312**	**312**	**312**	**312.0**	**312**	**312.0**	8.4	**312**	**312.0**	7.9	25
tai_5×5_7	303	**303**	308	**303**	**303.0**	**303**	**303.0**	7.4	**303**	**303.0**	7.3	25
tai_5×5_8	300	**300**	304	**300**	**300.0**	**300**	**300.0**	13.5	**300**	**300.0**	12.7	25
tai_5×5_9	353	**353**	358	**353**	**353.0**	**353**	**353.0**	11.5	**353**	**353.0**	7.9	25
tai_5×5_10	326	**326**	328	**326**	**326.0**	**326**	**326.0**	5.1	**326**	**326.0**	5.1	25
Average gap		0.000%	1.261%	0.000%	0.000%	0.000%	0.000%		0.000%	0.000%		
tai_7×7_1	435	**435**	436	**435**	**435.0**	**435**	**435.0**	6.6	**435**	**435.0**	2.9	49
tai_7×7_2	443	**443**	447	**443**	**443.0**	**443**	443.1	16.0	**443**	**443.0**	12.2	49
tai_7×7_3	468	**468**	472	**468**	**468.0**	**468**	468.1	16.8	**468**	**468.0**	9.2	49
tai_7×7_4	463	**463**	**463**	**463**	**463.0**	**463**	**463.0**	5.6	**463**	**463.0**	3.0	49
tai_7×7_5	416	**416**	417	**416**	**416.0**	**416**	**416.0**	2.8	**416**	**416.0**	2.9	49
tai_7×7_6	451	**451**	455	**451**	451.4	**451**	451.8	19.6	**451**	**451.0**	13.5	49
tai_7×7_7	422	**422**	426	**422**	422.2	**422**	422.5	10.5	**422**	**422.0**	13.6	49
tai_7×7_8	424	**424**	**424**	**424**	**424.0**	**424**	**424.0**	1.4	**424**	**424.0**	2.3	49
tai_7×7_9	458	**458**	**458**	**458**	**458.0**	**458**	**458.0**	0.6	**458**	**458.0**	1.3	49
tai_7×7_10	398	**398**	**398**	**398**	**398.0**	**398**	**398.0**	1.8	**398**	**398.0**	2.8	49
Average gap		0.000%	0.406%	0.000%	0.011%	0.000%	0.029%		0.000%	0.000%		
tai_10×10_1	637	**637**	**637**	**637**	637.4	**637**	639.1	18.2	**637**	**637.0**	9.4	50
tai_10×10_2	588	**588**	**588**	**588**	**588.0**	**588**	588.1	5.6	**588**	**588.0**	3.5	50
tai_10×10_3	598	**598**	**598**	**598**	**598.0**	**598**	599.8	22.9	**598**	**598.0**	10.1	50
tai_10×10_4	577	**577**	**577**	**577**	**577.0**	**577**	**577.0**	3.2	**577**	**577.0**	2.6	50
tai_10×10_5	640	**640**	**640**	**640**	**640.0**	**640**	640.9	17.4	**640**	**640.0**	4.0	50
tai_10×10_6	538	**538**	**538**	**538**	**538.0**	**538**	**538.0**	0.5	**538**	**538.0**	1.1	50
tai_10×10_7	616	**616**	**616**	**616**	**616.0**	**616**	616.3	11.4	**616**	**616.0**	3.9	50
tai_10×10_8	595	**595**	**595**	**595**	**595.0**	**595**	595.6	17.5	**595**	**595.0**	7.0	50
tai_10×10_9	595	**595**	**595**	**595**	**595.0**	**595**	**595.0**	11.7	**595**	**595.0**	4.1	50
tai_10×10_10	596	**596**	**596**	**596**	**596.0**	**596**	596.1	9.8	**596**	**596.0**	5.0	50
Average gap		0.000%	0.000%	0.000%	0.005%	0.000%	0.091%		0.000%	0.000%		

Table 6.2 (continued)

Problem	BKS	GA-Liaw	GA-Prins	Beam-ACO		PSO-mP-ASG2			PSO-mP-ASG2+BS			Time
				Best	Average	Best	Average	*t*	Best	Average	*t*	limit(s)
tai_15×15_1	937	**937**	**937**	**937**	**937.0**	**937**	**937.0**	4.6	**937**	**937.0**	4.3	112.5
tai_15×15_2	918	**918**	**918**	**918**	**918.0**	**918**	918.9	20.5	**918**	**918.0**	9.1	112.5
tai_15×15_3	871	**871**	**871**	**871**	**871.0**	**871**	**871.0**	5.0	**871**	**871.0**	4.3	112.5
tai_15×15_4	934	**934**	**934**	**934**	**934.0**	**934**	**934.0**	3.2	**934**	**934.0**	3.9	112.5
tai_15×15_5	946	**946**	**946**	**946**	**946.0**	**946**	946.4	16.1	**946**	**946.0**	5.7	112.5
tai_15×15_6	933	**933**	**933**	**933**	**933.0**	**933**	**933.0**	10.0	**933**	**933.0**	4.7	112.5
tai_15×15_7	891	**891**	**891**	**891**	**891.0**	**891**	891.6	22.3	**891**	**891.0**	10.4	112.5
tai_15×15_8	893	**893**	**893**	**893**	**893.0**	**893**	**893.0**	1.2	**893**	**893.0**	17.3	112.5
tai_15×15_9	899	**899**	**899**	**899**	899.7	**899**	903.3	46.6	**899**	899.2	26.6	112.5
tai_15×15_10	902	**902**	**902**	**902**	**902.0**	**902**	903.0	12.7	**902**	**902.0**	6.9	112.5
Average gap		0.000%	0.000%	0.000%	0.007%	0.000%	0.079%		0.000%	0.002%		
tai_20×20_1	1155	**1155**	**1155**	**1155**	**1155.0**	**1155**	1155.8	24.9	**1155**	**1155.0**	16.6	200
tai_20×20_2	1241	**1241**	**1241**	**1241**	**1241.0**	1242	1245.2	56.8	**1241**	**1241.0**	23.5	200
tai_20×20_3	1257	**1257**	**1257**	**1257**	**1257.0**	**1257**	**1257.0**	5.5	**1257**	**1257.0**	19.6	200
tai_20×20_4	1248	**1248**	**1248**	**1248**	**1248.0**	**1248**	**1248.0**	4.0	**1248**	**1248.0**	19.6	200
tai_20×20_5	1256	**1256**	**1256**	**1256**	**1256.0**	**1256**	**1256.0**	7.9	**1256**	**1256.0**	19.6	200
tai_20×20_6	1204	**1204**	**1204**	**1204**	**1204.0**	**1204**	1204.1	12.0	**1204**	**1204.0**	19.6	200
tai_20×20_7	1294	**1294**	**1294**	**1294**	**1294.0**	**1294**	1296.6	48.5	**1294**	**1294.0**	25.4	200
tai_20×20_8	1169	1177	1171	**1169**	1170.3	1173	1177.6	80.9	**1169**	1170.0	50.9	200
tai_20×20_9	1289	**1289**	**1289**	**1289**	**1289.0**	**1289**	**1289.0**	2.7	**1289**	**1289.0**	78.2	200
tai_20×20_10	1241	**1241**	**1241**	**1241**	**1241.0**	**1241**	**1241.0**	1.2	**1241**	**1241.0**	78.2	200
Average gap		0.068%	0.017%	0.000%	0.011%	0.042%	0.134%		0.000%	0.008%		

Table 6.3 Results of the test problems proposed by Brucker et al. (1997)

Problem	BKS	GA-Liaw	GA-Prins	Beam-ACO		PSO-mP-ASG2			PSO-mP-ASG2+BS			Time
				Best	Average	Best	Average	*t*	Best	Average	*t*	limit(s)
j5-per0-0	1042	**1042**	1050	**1042**	**1042.0**	**1042**	**1042.0**	6.4	**1042**	**1042.0**	6.3	125
j5-per0-1	1054	**1054**	**1054**	**1054**	**1054.0**	**1054**	**1054.0**	0.0	**1054**	**1054.0**	1.3	125
j5-per0-2	1063	**1063**	1085	**1063**	**1063.0**	**1063**	**1063.0**	24.6	**1063**	**1063.0**	24.6	125
j5-per10-0	1004	**1004**	**1004**	**1004**	**1004.0**	**1004**	**1004.0**	34.9	**1004**	**1004.0**	34.8	125
j5-per10-1	1002	**1002**	**1002**	**1002**	**1002.0**	**1002**	**1002.0**	25.3	**1002**	**1002.0**	25.2	125
j5-per10-2	1006	**1006**	**1006**	**1006**	**1006.0**	**1006**	**1006.0**	25.2	**1006**	**1006.0**	25.1	125
j5-per20-0	1000	**1000**	1004	**1000**	**1000.0**	**1000**	**1000.0**	19.1	**1000**	**1000.0**	17.6	125
j5-per20-1	1000	**1000**	**1000**	**1000**	**1000.0**	**1000**	**1000.0**	1.0	**1000**	**1000.0**	2.7	125
j5-per20-2	1012	**1012**	**1012**	**1012**	**1012.0**	**1012**	**1012.0**	0.0	**1012**	**1012.0**	0.0	125
Average gap		0.000%	0.360%	0.000%	0.000%	0.000%	0.000%		0.000%	0.000%		
j6-per0-0	1056	**1056**	1080	**1056**	**1056.0**	**1056**	**1056.0**	37.2	**1056**	**1056.0**	42.1	180
j6-per0-1	1045	**1045**	**1045**	**1045**	1049.7	**1045**	**1045.0**	51.1	**1045**	**1045.0**	59.7	180
j6-per0-2	1063	**1063**	1079	**1063**	**1063.0**	**1063**	**1063.0**	70.5	**1063**	**1063.0**	72.6	180
j6-per10-0	1005	**1005**	1016	**1005**	**1005.0**	**1005**	**1005.0**	41.0	**1005**	**1005.0**	45.5	180
j6-per10-1	1021	**1021**	1036	**1021**	**1021.0**	**1021**	**1021.0**	20.9	**1021**	**1021.0**	21.0	180
j6-per10-2	1012	**1012**	**1012**	**1012**	**1012.0**	**1012**	**1012.0**	8.2	**1012**	**1012.0**	8.5	180
j6-per20-0	1000	**1000**	1018	**1000**	1003.6	**1000**	1000.6	98.4	**1000**	**1000.0**	77.5	180
j6-per20-1	1000	**1000**	**1000**	**1000**	**1000.0**	**1000**	**1000.0**	0.4	**1000**	**1000.0**	1.5	180
j6-per20-2	1000	**1000**	1001	**1000**	**1000.0**	**1000**	**1000.0**	29.1	**1000**	**1000.0**	30.6	180
Average gap		0.000%	0.916%	0.000%	0.090%	0.000%	0.007%		0.000%	0.000%		
j7-per0-0	(1048)	1063	1071	**1048**	1052.7	1052	1053.8	94.5	1050	1051.2	104.9	245
j7-per 0-1	1055	1058	1076	1057	1057.8	**1055**	1061.0	126.8	1057	1058.8	155.8	245
j7-per 0-2	1056	1059	1082	1058	1059.0	**1056**	1059.7	97.1	**1056**	1057.0	124.5	245
j7-per10-0	1013	1022	1036	**1013**	1016.7	1016	1018.3	103.3	**1013**	1016.1	183.8	245
j7-per10-1	1000	1014	1010	**1000**	1002.5	**1000**	**1000.0**	76.6	**1000**	**1000.0**	81.9	245
j7-per10-2	1011	1020	1035	1016	1019.4	1013	1019.5	71.4	1013	1014.9	125.6	245
j7-per20-0	1000	**1000**	**1000**	**1000**	**1000.0**	**1000**	**1000.0**	0.3	**1000**	**1000.0**	1.9	245
j7-per20-1	1005	1011	1030	**1005**	1007.6	1007	1008.1	148.7	1007	1008.0	143.2	245
j7-per20-2	1003	1010	1020	**1003**	1007.3	**1003**	1004.1	134.1	**1003**	1004.7	160.9	245
Average gap		0.719%	1.830%	0.097%	0.346%	0.119%	0.360%		0.086%	0.211%		
j8-per0-1	(1039)	-	1075	**1039**	1048.7	1045	1051.7	147.4	**1039**	1043.3	220.8	320
j8-per 0-2	(1052)	-	1073	**1052**	1057.1	**1052**	1058.3	244.6	**1052**	1053.6	271.9	320
j8-per10-0	(1020)	-	1053	**1020**	1026.9	1024	1033.1	181.6	**1020**	1026.1	205.0	320
j8-per10-1	(1002)*	-	1029	1004	1012.4	1008	1018.3	70.7	**1002**	1007.6	202.2	320
j8-per10-2	(1002)*	-	1027	1009	1013.7	**1002**	1012.5	137.8	**1002**	1006.0	162.8	320
j8-per20-0	1000	-	1015	**1000**	1001.0	**1000**	1000.9	158.9	**1000**	1000.6	136.9	320
j8-per20-1	1000	-	**1000**	**1000**	**1000.0**	**1000**	**1000.0**	1.3	**1000**	**1000.0**	4.5	320
j8-per20-2	1000	-	1014	**1000**	1000.6	**1000**	1003.2	97.9	**1000**	**1000.0**	105.8	320
Average gap			2.098%	0.112%	0.555%	0.196%	0.771%		0.000%	0.271%		

*The BKS is found by one of our PSOs.

Table 6.4 shows the results of the test problems proposed by Guéret and Prins (1999). In this problem set, PSO-mP-ASG2 performs better than other algorithms and obtained 16 new best-known solutions. The PSO hybridized with beam search does not perform well, because the instances from (Guéret & Prins, 1999) are designed to maximize a new lower bound greater than the traditional one. Therefore, the traditional lower bound calculated by equations (5.6), (5.7), and (5.8) is hard to discriminate in the good and bad partial solutions in this problem set (Guéret & Prins, 1999).

Consequently, better solutions can be obtained when the PSO performs much more iterations instead of spending computation time on an inefficient beam search. However, it is evident that both the PSOs outperform Beam-ACO in this problem set.

Table 6.4 Results of the test problems proposed by Guéret and Prins (1999)

Problem	BKS	GA-Prins	Beam-ACO		PSO-mP-ASG2			PSO-mP-ASG2+BS			Time
			Best	Average	Best	Average	*t*	Best	Average	*t*	limit(s)
gp03-01	1168	**1168**	**1168**	**1168.0**	**1168**	**1168.0**	0.0	**1168**	**1168.0**	0.0	45
gp03-02	1170	**1170**	**1170**	**1170.0**	**1170**	**1170.0**	0.0	**1170**	**1170.0**	0.0	45
gp03-03	1168	**1168**	**1168**	**1168.0**	**1168**	**1168.0**	0.0	**1168**	**1168.0**	0.0	45
gp03-04	1166	**1166**	**1166**	**1166.0**	**1166**	**1166.0**	0.0	**1166**	**1166.0**	0.0	45
gp03-05	1170	**1170**	**1170**	**1170.0**	**1170**	**1170.0**	0.0	**1170**	**1170.0**	0.0	45
gp03-06	1169	**1169**	**1169**	**1169.0**	**1169**	**1169.0**	0.0	**1169**	**1169.0**	0.0	45
gp03-07	1165	**1165**	**1165**	**1165.0**	**1165**	**1165.0**	0.0	**1165**	**1165.0**	0.0	45
gp03-08	1167	**1167**	**1167**	**1167.0**	**1167**	**1167.0**	0.0	**1167**	**1167.0**	0.0	45
gp03-09	1162	**1162**	**1162**	**1162.0**	**1162**	**1162.0**	0.0	**1162**	**1162.0**	0.0	45
gp03-10	1165	**1165**	**1165**	**1165.0**	**1165**	**1165.0**	0.0	**1165**	**1165.0**	0.0	45
Average gap		0.000%	0.000%	0.000%	0.000%	0.000%		0.000%	0.000%		
gp04-01	1281	**1281**	**1281**	**1281.0**	**1281**	**1281.0**	0.0	**1281**	**1281.0**	0.0	80
gp04-02	1270	**1270**	**1270**	**1270.0**	**1270**	**1270.0**	0.3	**1270**	**1270.0**	0.3	80
gp04-03	1288	**1288**	**1288**	**1288.0**	**1288**	**1288.0**	0.3	**1288**	**1288.0**	0.3	80
gp04-04	1261	**1261**	**1261**	**1261.0**	**1261**	**1261.0**	30.0	**1261**	**1261.0**	30.0	80
gp04-05	1289	**1289**	**1289**	**1289.0**	**1289**	**1289.0**	30.0	**1289**	**1289.0**	30.0	80
gp04-06	1269	**1269**	**1269**	**1269.0**	**1269**	**1269.0**	30.0	**1269**	**1269.0**	30.0	80
gp04-07	1267	**1267**	**1267**	**1267.0**	**1267**	**1267.0**	30.2	**1267**	**1267.0**	30.2	80
gp04-08	1259	**1259**	**1259**	**1259.0**	**1259**	**1259.0**	22.7	**1259**	**1259.0**	30.2	80
gp04-09	1280	**1280**	**1280**	**1280.0**	**1280**	**1280.0**	2.9	**1280**	**1280.0**	2.9	80
gp04-10	1263	**1263**	**1263**	**1263.0**	**1263**	**1263.0**	2.9	**1263**	**1263.0**	2.9	80
Average gap		0.000%	0.000%	0.000%	0.000%	0.000%		0.000%	0.000%		
gp05-01	1245	**1245**	**1245**	**1245.0**	**1245**	**1245.0**	0.0	**1245**	**1245.0**	0.2	125
gp05-02	1247	**1247**	**1247**	**1247.0**	**1247**	**1247.0**	0.0	**1247**	**1247.0**	0.0	125
gp05-03	1265	**1265**	**1265**	**1265.0**	**1265**	**1265.0**	0.0	**1265**	**1265.0**	0.1	125
gp05-04	1258	**1258**	**1258**	1258.6	**1258**	**1258.0**	104.6	**1258**	**1258.0**	105.2	125
gp05-05	1280	**1280**	**1280**	**1280.0**	**1280**	**1280.0**	10.5	**1280**	**1280.0**	31.5	125
gp05-06	1269	**1269**	**1269**	1269.1	**1269**	**1269.0**	18.0	**1269**	**1269.0**	18.0	125
gp05-07	1269	**1269**	**1269**	**1269.0**	**1269**	**1269.0**	0.0	**1269**	**1269.0**	2.7	125
gp05-08	1287	**1287**	**1287**	**1287.0**	**1287**	**1287.0**	0.0	**1287**	**1287.0**	0.0	125
gp05-09	1262	**1262**	**1262**	**1262.0**	**1262**	**1262.0**	22.0	**1262**	**1262.0**	22.0	125
gp05-10	1254	**1254**	**1254**	1254.6	**1254**	**1254.0**	63.0	**1254**	**1254.0**	63.5	125
Average gap		0.000%	0.000%	0.010%	0.000%	0.000%		0.000%	0.000%		
gp06-01	1264	**1264**	**1264**	1264.7	**1264**	**1264.0**	176.3	**1264**	**1264.0**	176.1	180
gp06-02	1285	**1285**	**1285**	1285.7	**1285**	**1285.0**	140.0	**1285**	**1285.0**	147.8	180
gp06-03	(1255)	**1255**	**1255**	**1255.0**	**1255**	1255.2	166.7	**1255**	1255.6	133.1	180
gp06-04	1275	**1275**	**1275**	**1275.0**	**1275**	**1275.0**	60.1	**1275**	**1275.0**	60.8	180
gp06-05	1299	1300	**1299**	1299.2	**1299**	**1299.0**	159.6	**1299**	**1299.0**	159.6	180
gp06-06	1284	**1284**	**1284**	**1284.0**	**1284**	**1284.0**	96.6	**1284**	**1284.0**	109.4	180
gp06-07	(1290)	**1290**	**1290**	**1290.0**	**1290**	**1290.0**	0.4	**1290**	**1290.0**	1.6	180
gp06-08	1265	1266	**1265**	1265.2	**1265**	1265.2	151.9	**1265**	1265.5	134.3	180
gp06-09	(1243)	**1243**	**1243**	**1243.0**	**1243**	**1243.0**	141.6	**1243**	1243.1	156.5	180
gp06-10	(1254)	**1254**	**1254**	**1254.0**	**1254**	**1254.0**	80.5	**1254**	**1254.0**	79.8	180
Average gap		0.016%	0.000%	0.013%	0.000%	0.003%		0.000%	0.009%		

Table 6.4 (*continued*)

Problem	BKS	GA-Prins	Beam-ACO		PSO-mP-ASG2			PSO-mP-ASG2+BS			Time
			Best	Average	Best	Average	*t*	Best	Average	*t*	limit(s)
gp07-01	(1159)	**1159**	**1159**	**1159.0**	**1159**	**1159.0**	180.5	**1159**	1159.3	223.7	245
gp07-02	(1185)	**1185**	**1185**	**1185.0**	**1185**	**1185.0**	0.3	**1185**	**1185.0**	1.2	245
gp07-03	1237	**1237**	**1237**	**1237.0**	**1237**	**1237.0**	3.9	**1237**	**1237.0**	9.5	245
gp07-04	(1167)	**1167**	**1167**	**1167.0**	**1167**	**1167.0**	188.2	**1167**	**1167.0**	160.4	245
gp07-05	1157	**1157**	**1157**	**1157.0**	**1157**	**1157.0**	130.3	**1157**	**1157.0**	139.1	245
gp07-06	(1193)	**1193**	**1193**	1193.9	**1193**	**1193.0**	153.1	**1193**	1193.1	198.6	245
gp07-07	1185	**1185**	**1185**	1185.1	**1185**	**1185.0**	0.4	**1185**	**1185.0**	1.4	245
gp07-08	(1180)	1181	**1180**	1181.4	**1180**	**1180.0**	139.1	**1180**	**1180.0**	139.4	245
gp07-09	(1220)	**1220**	**1220**	1220.1	**1220**	**1220.0**	123.6	**1220**	**1220.0**	143.9	245
gp07-10	1270	**1270**	**1270**	1270.1	**1270**	**1270.0**	0.1	**1270**	**1270.0**	0.5	245
Average gap		0.008%	0.000%	0.020%	0.000%	0.000%		0.000%	0.003%		
gp08-01	1130	1160	**1130**	1132.4	**1130**	1143.5	204.3	1133	1140.3	277.3	320
gp08-02	(1135)	1136	**1135**	1136.1	**1135**	**1135.0**	192.5	**1135**	1135.4	258.3	320
gp08-03	1110	1111	1111	1113.7	**1110**	1110.8	229.0	**1110**	1114.0	240.3	320
gp08-04	(1153)*	1168	1154	1156.0	**1153**	**1153.0**	306.3	**1153**	1153.2	308.1	320
gp08-05	1218	**1218**	1219	1219.8	**1218**	**1218.0**	297.2	**1218**	1218.9	56.6	320
gp08-06	(1115)*	1128	1116	1123.2	**1115**	1117.2	224.7	**1115**	1126.9	249.6	320
gp08-07	(1126)	1128	**1126**	1134.6	**1126**	1127.7	288.6	**1126**	1129.8	287.3	320
gp08-08	(1148)	**1148**	**1148**	1149.0	**1148**	**1148.0**	133.1	**1148**	**1148.0**	179.3	320
gp08-09	1114	1120	1117	1119.0	**1114**	**1114.0**	156.0	**1114**	1114.3	223.6	320
gp08-10	(1161)	**1161**	**1161**	1161.5	**1161**	1161.1	289.3	**1161**	1161.4	217.1	320
Average gap		0.602%	0.062%	0.311%	0.000%	0.161%		0.027%	0.284%		
gp09-01	(1129)*	1143	1135	1142.8	**1129**	1129.4	363.0	**1129**	1133.2	376.3	405
gp09-02	(1110)*	1114	1112	1113.7	**1110**	1111.8	284.2	1112	1114.1	335.9	405
gp09-03	(1116)*	1118	1118	1120.4	**1116**	1117.0	235.2	1117	1117.0	313.4	405
gp09-04	1130	1131	**1130**	1140.0	**1130**	1130.7	355.5	**1130**	1135.8	328.7	405
gp09-05	1180	**1180**	**1180**	1180.5	**1180**	**1180.0**	6.6	**1180**	**1180.0**	22.3	405
gp09-06	(1093)	1117	**1093**	1195.6	**1093**	1095.1	319.0	**1093**	1094.1	277.2	405
gp09-07	(1091)*	1119	1097	1101.4	1094	1096.7	344.1	**1091**	1096.5	376.4	405
gp09-08	(1106)	1110	**1106**	1113.7	1108	1108.0	186.8	1108	1108.3	334.6	405
gp09-09	(1123)*	1132	1127	1132.5	**1123**	1124.6	219.1	1126	1126.5	358.6	405
gp09-10	(1112)*	1130	1120	1126.3	**1112**	1124.8	214.1	1122	1126.5	297.7	405
Average gap		0.941%	0.252%	1.601%	0.046%	0.252%		0.162%	0.374%		
gp10-01	(1093)*	1113	1099	1109.0	**1093**	1097.4	396.3	**1093**	1096.8	455.7	500
gp10-02	(1097)*	1120	1101	1107.4	**1097**	**1097.0**	348.7	**1097**	1099.1	382.7	500
gp10-03	(1081)	1101	1082	1098.0	**1081**	1087.1	404.6	1084	1090.3	450.8	500
gp10-04	(1083)*	1090	1093	1096.6	1086	1089.1	294.0	**1083**	1092.1	371.8	500
gp10-05	(1073)*	1094	1083	1092.4	**1073**	1086.5	288.1	1082	1092.2	314.1	500
gp10-06	(1071)*	1074	1088	1104.6	**1071**	**1071.0**	148.5	**1071**	1074.3	289.7	500
gp10-07	(1080)*	1083	1084	1091.5	**1080**	1081.0	133.0	1081	1081.1	167.4	500
gp10-08	(1095)*	1098	1099	1104.8	**1095**	1097.3	358.8	1097	1097.6	324.5	500
gp10-09	(1115)*	1121	1121	1128.7	**1115**	1117.8	357.5	1123	1127.0	428.2	500
gp10-10	(1092)	1095	1097	1106.7	**1092**	1092.2	479.3	**1092**	1094.0	487.9	500
Average gap		1.002%	0.618%	1.470%	0.028%	0.335%		0.211%	0.590%		

*The BKS is found by one of our PSOs.

6.7 Concluding Remarks

We have presented a PSO for open shop scheduling problems in this chapter. We modified the representation of particle position, particle movement, and particle velocity to better suit it for OSSP. We also proposed a new decoding operator (mP-ASG), which decodes particle positions into parameterized active schedules. Furthermore, we added fathoming constraints to mP-ASG and then hybridized it with beam search. The computational results show that our PSO can obtain many new best-known solutions of the test problems.

For further research, we will try to apply our PSO to other shop scheduling problems. In addition, further research topics include how to modify the particle position representation, particle movement, and particle velocity to better suit them to the problem. Table 6.5 shows the summary of the PSO for OSSP.

Table 6.5 Summary of the PSO for OSSP

		Components	The concept of this components
1	Particle Position Representation	Priority weights	We represent the preference-list by priorities, which can save computation time when we implement insert operator.
2	Particle Velocity	Inertia	Because there is no preference constraint between the operations of a job, if we swap two operations at the same time, the new solution will be much different than the original one and lose the correlation property. Therefore, we implement the insert operator, just move one operation at a time, and earn more correlation property.
	Particle Movement	Insert operator	
3	Decoding Operator	G&T algorithm mP-ASG Beam search	The mP-ASG can much restrict the search area but not exclude the optimal solution.
4	Other Strategies	Diversification	The diversification strategy can prevent particles rapped in local optima. We do not implement the local search strategy, because the neighborhood size of OSSP is huge and the local search strategy is in efficient.

6.8 Appendix

A pseudo code of the PSO for OSSP is given below:

// ↓ initializing

Initialize a population of particles with random positions.

for each particle *k* ***do***

Decode X^k (the position of particle *k*) into a schedule S^k.

Set the *k*th *pbest* solution ($pbest^k$) equal to S^k, $pbest^k \leftarrow S^k$.

end for

Set *gbest* solution equal to the best $pbest^k$.

// ↑ initializing

repeat

Update velocities according to Figure 6.1.

for each particle *k* ***do***

Move particle *k* according to Figure 6.2.

Decode X^k into S^k.

Update *pbest* solutions and *gbest* solution according to Figure 5.6.

end for

until maximum CPU time limit is reached.

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Conclusions

The original PSO is used to solve continuous optimization problems. Due to solution spaces of discrete optimization problems are discrete, we have to develop new PSO designs to better suit it for discrete optimization problems. The contribution of this research is that we proposed several PSO designs for discrete optimization problems. The new PSO designs are better suit for discrete optimization problems, and differ from the original PSO. In this research, we separated a PSO design into five parts: particle position representation, particle velocity, particle movement, decoding operator, and other search strategies. We can develop a new PSO design by redesign these five parts.

In chapter 4, we presented a binary PSO for the multidimensional 0-1 knapsack problem (MKP). This PSO design focuses on the concept of building blocks, which is the basis of another evolution computation algorithm—genetic algorithm. The particle velocity is represented by blocks. Particles obtain new blocks from *gbest* and *pbest* solutions. Moreover, the selection strategy can recognize superior blocks and and accumulate the superior blocks in the swarm. The computational results show that the concept of building blocks works in PSO design. Therefore, we can design other new PSOs based on the concept of building blocks.

In chapter 5, we presented a PSO for the job shop scheduling problem (JSSP). This PSO design focuses on the particle position representation. In this PSO, the particle position is represented by preference list-based representation, which has more Lamarckian than the original PSO design—priority based representation. We also tested and compared these two particle position representations. The computational results show that the preference list-based representation we proposed outperforms the priority based representation. It also demonstrated that the particle position representation with more Lamarckian performs better. Therefore, we can design new particle position representation based on Lamarckian property in further researches.

In chapter 6, we presented a PSO for the open shop scheduling problem (OSSP). This PSO design focuses on comparing decoding operators. In this PSO, we implemented a modified parameterized active schedule generation algorithm (mP-ASG), which decodes particle positions into parameterized active schedules. In mP-ASG, we can reduce or increase the search area between non-delay schedules and active schedules by controlling the maximum delay time allowed. Furthermore, we added fathoming constraints to mP-ASG and then hybridized it with beam search. The computational results show that a decoding operator, which can map the positions to the solution space in a smaller region but not excluding the optimal solution, is performs better.

7.2 Future Works

There are two aspects for further research: (1) new PSO designs, and (2) other applications. We described some principles for new PSO designs in chapter 3. In the further research, we can develop new PSO designs by these principles and find out new design principles at the same time.

On the other hand, we can also implement the new PSO designs to other combinatorial optimization problems for example: assignment problems, network problems, multiobject combinatorial optimization problems…etc.

References

Adams, J., Balas, E., & Zawack, D. (1988). The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34(3), 391-401.

Alcaide, D., Sicilia, J., & Vigo, D. (1997). Heuristic approaches for the minimum makespan open shop problem. *Trabajos de Investigación Operativa*, 5(2), 283-296.

Allahverdi, A, & Al-Anzi, F.S. (2006). A PSO and a tabu search heuristics for the assembly scheduling problem of the two-stage distributed database application. *Computers and Operations Research*, 33(4), 1056-1080.

Angeline, P.J. (1998). Using selection to improve particle swarm optimization. *Proceedings of the IEEE International Conference on Evolutionary Computation*, 84–89.

Bagchi, T.P. (1999). *Multiobjective scheduling by genetic algorithms*. Kluwer Academic Publishers.

Balas, E., & Vazacopoulos, A. (1998). Guided local search with shifting bottleneck for job shop scheduling. *Management Science*, 44(2), 262-275.

Bean, J. (1994). Genetic algorithms and random keys for sequencing and optimization. *Operations Research Society of America (ORSA) Journal on Computing*, 6, 154-160.

Beasley J.E. (1990). OR-Library: distributing test problems by electronic mail. Journal o*f the Operational Research Society*, 14, 1069-1072.

Blum, C. (2005). Beam-ACO—hybridizing ant colony optimization with beam search: and application to open shop scheduling. *Computers & Operations Research*, 32, 1565-1591.

Brucker, P., Hurink, J., Jurisch, B., & Wöstmann, B. (1997). A branch & bound algorithm for the open-shop problem. *Discrete Applied Mathematics*, 76, 43-59.

Chu, P.C., & Beasley, J.E. (1998). A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristic*, 4, 63–86.

Colak, S., & Agarwal, A. (2005). Non-greedy heuristics and augmented neural networks for the open-shop scheduling problem. *Naval Research Logistics*, 52, 631-644.

Davis, L. (1985). Job shop scheduling with genetic algorithm. *Proceedings of the first International Conference on Genetic Algorithms*, 163-140.

Dorndorf, U., Pesch, E., & Phan-Huy, T. (2001). Solving the open-shop scheduling problem. *Journal of scheduling*, 4(3), 157-174.

Drexl, A. (1988). A simulated *annealing* approach to the multiconstraint zero-one knapsack problem. *Computing*, 40, 1–8.

Eberhart R., & Shi Y. (2001). Particle swarm optimization: developments, applications and resources. *Proceedings of the 2001 IEEE Congress on Evolutionary Computation*, 81-86.

Fisher, H., & Thompson, G.L. (1963). *Industrial Scheduling*. Englewood Cliffs, NJ: Prentice-Hall.

French, S. (1982). *Sequencing and scheduling: an introduction to the mathematics of the job-shop*. UK: Horwood.

Fréville, A. (2004). The multidimensional 0–1 knapsack problem: An overview. *European Journal of Operational Research*, 155(1), 1–21.

Fréville, A., & Hanafi, S. (2005). The multidimensional 0-1 knapsack problem — bounds and *computational* aspects. *Annals of Operations Research*, 139, 195–227.

Garey, M.R., Johnson, D.S., & Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1, 117-129.

Gen, M., & Cheng, R. (1997). *Genetic algorithms and engineering design*. New York: Wiley.

Giffler, J., & Thompson, G.L. (1960). Algorithms for solving production scheduling problems. *Operations Research*, 8, 487-503.

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

Glover, F. (1989). Tabu search: Part I. *ORSA Journal on Computing*, 1, 190-206.

Glover, F. (1990). Tabu search: Part II. *ORSA Journal on Computing*, 2, 4-32.

Glover, F., & Kochenberger G.A. (1996). Critical event tabu search for multidimensional knapsack problems, in I.H. Osman, and J.P. Kelly, (Eds.), *Metaheuristics: The Theory and Applications*, Kluwer Academic Publishers, Dordrecht, pp.407–427.

Goldberg, D.E. (2002). *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*. Kluwer Academic Publishers, Dordrecht.

Gonçalves, J.F., & Beirão, N.C., (1999). Um Algoritmo Genético Baseado em Chaves Aleatórias para Sequenciamento de Operações. *Revista Associação Portuguesa de Desenvolvimento e Investigação Operacional* 19, 123-137 (in Portuguese).

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(1), 77-95.

Gonzalez, T., & Sahni, S. (1976). Open shop scheduling to minimize finish time. *Journal of the ACM*, 23(4), 665-679.

Guéret, C., & Prins, C. (1998). Classical and new heuristics for the open-shop problem: a computational evaluation, *European Journal of Operational Research*, 107(2), 306-314.

Guéret, C., & Prins, C. (1999). A new lower bound for the open-shop problem. *Annals of Operations Research*, 92, 165-183.

Hanafi, S., & Fréville, A. (1998). An efficient tabu search approach for the 0-1 multidimensional knapsack problem. *European Journal of Operational Research*, 106, 659–675.

Jain, A.S., Rangaswamy, B., & Meeran, S. (2000). New and “stronger” job-shop neighbourhoods: a focus on the method of Nowicki and Smutnicki (1996). *Journal of Heuristics*, 6, 457-480.

Jin, Y.X., Cheng, H.Z., Yan, J.Y., Zhang, L. (2007). New discrete method for particle swarm optimization and its application in transmission network expansion planning. *Electric Power Systems Research*, 77(3-4), 227-233.

Kennedy, J., & Eberhart, R.C. (1995). Particle swarm optimization. *Proceedings of the 1995 IEEE International Conference on Neural Networks*, 4, 1942-1948.

Kennedy, J., & Eberhart, R.C. (1997). A discrete binary version of the particle swarm algorithm. *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, 5, 4104–4108.

Kobayashi, S., Ono, I., & Yamamura, M. (1995). An efficient genetic algorithm for job shop scheduling problems. *Proceedings of the Sixth International*

Conference on Genetic Algorithms, 506-511.

Lawrence, S., (1984). Resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques. Graduate School of Industrial Administration (GSIA), Carnegie Mellon University, Pittsburgh, PA.

Li, N., Liu, F., Sun, D., & Huang, C. (2004). Particle Swarm Optimization for Constrained Layout Optimization. *Proceedings of Fifth World Congress on Intelligent Control and Automation (WCICA 2004)*, 3, 2214-2218.

Lian, Z., Gu, X., & Jiao, B. (2006). A similar particle swarm optimization algorithm for permutation flowshop scheduling to minimize makespan. *Applied Mathematics and Computation*,175(1), 773-785.

Liao, C.J., Tseng, C.T., & Pin, L. (2007). A discrete version of particle swarm optimization for flowshop scheduling problems. *Computers and Operations Research*, 34(10), 3099-3111.

Liaw, C-F. (1999a). Applying simulated annealing to the open shop scheduling problem. *IIE Transactions*, 31, 457-465.

Liaw, C-F. (1999b). A tabu search algorithm for the open shop scheduling problem. *Computers & Operations Research*, 26, 109-126.

Liaw, C-F (2000). A hybrid genetic algorithm for the open shop scheduling problem. *European Journal of Operational Research*, 124, 28-42.

Lourenço, H.R. (1995). Local optimization and the job-shop scheduling problem. *European Journal of Operational Research*, 83, 347-364.

Mattfeld, D.C. (1996). *Evolutionary Search and the Job Shop: Investigations on Genetic Algorithms for Production Scheduling*. Physica-Verlag, Heidelberg, Germany.

Nowicki, E., & Smutnicki, C. (1996). A fast taboo search algorithm for the job shop problem. *Management Science*, 42(6), 797-813.

Osorio, M.A., Glover, F., & Hammer, P. (2002). Cutting and surrogate constraint analysis for improved multidimensional knapsack solutions. *Annals of Operations Research*, 117, 71–93.

Ow, P.S., & Morton, T.E. (1988). Filtered beam search in scheduling. *International Journal of Production Research*, 26, 297-307.

Pang, W., Wang, K.P., Zhou, C.G.., Dong, L.J., Liu, M., Zhang, H.Y., & Wang, J.Y.

(2004). Modified particle swarm optimization based on space transformation for solving traveling salesman problem. *Proceedings of 2004 International Conference on Machine Learning and Cybernetics*, 4, 2342-2346.

Pezzella, F., & Merelli, E. (2000). A tabu search method guided by shifting bottleneck for the job shop scheduling problem. *European Journal of Operational Research*, 120(2), 297-310.

Pirkul, H. (1987). A heuristic solution procedure for the multiconstraint zero-one knapsack problem. *Naval Research Logistics*, 34, 161–172.

Prins, C. (2000). Competitive genetic algorithms for the open-shop scheduling problem. *Mathematical Methods of Operations Research*, 52, 389-411.

Rastegar, R., Meybodi, M.R., & Badie, K. (2004). A new discrete binary particle swarm optimization based on learning automata. *Proceedings of the IEEE International Conference on Machine Learning and Applications*, 456–465.

Salman, A., Ahmad, I., & Al-Madani, S. (2002). Particle swarm optimization for task assignment problem. *Microprocessors and Microsystems*, 26(8), 363-371.

Sha, D.Y., & C.-Y. Hsu. (2006). "A modified parameterized active schedule generation algorithm for the job shop scheduling problem," *Proceedings of the 36th International Conference on Computers and Industrial Engineering (ICCIE 2006)*, pp. 702-712, Taiwan, ROC.

Shi Y, & Eberhart R.C. (1998a). Parameter selection in particle swarm optimization. *Proceedings of the 7th International Conference on Evolutionary Programming*, 591-600.

Shi Y, & Eberhart R.C. (1998b). A modified particle swarm optimizer. *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation*, 69-73.

Stacey, A., Jancic, M., & Grundy, I. (2003). Particle swarm optimization with mutation. *Proceedings of the 2003 IEEE Congress on Evolutionary Computation*, 2, 1425-1430.

Sun, D., Batta, R., & Lin, L. (1995). Effective job shop scheduling through active chain manipulation. *Computers & Operations Research*, 22(2), 159-172.

Taillard, E.D. (1993), Benchmarks for basic scheduling problems, *European Journal of Operational Research*, 64, 278-285.

Tasgetiren, M.F., Liang, Y-C., Sevkli, M., & Gencyilmaz, G. (2007). A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing. *European Journal of Operational Research*, 177(3), 1930-1947.

Vasquez, M., & Hao, J.K. (2001). A hybrid approach for the 0-1 multidimensional knapsack problem. *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, 1, 328–333.

Vasquez, M., & Vimont, Y. (2005). Improved results on the 0-1 multidimensional knapsack problem. *European Journal of Operational Research*, 165, 70–81.

Wang, K.P., Huang, L., Zhou, C.G., & Pang, W. (2003). Particle swarm optimization for traveling salesman problem. *Proceedings of International Conference on Machine Learning and Cybernetics*, 3, 1583-1585.

Wang, L., & Zheng, D. (2001). An effective hybrid optimization strategy for job-shop scheduling problems. *Computers & Operations Research*, 28, 585-596.

Wu, B., Zhao, Y., Ma Y., Dong, H., & Wang, W. (2004). Particle Swarm Optimization method for Vehicle Routing Problem. *Proceedings of Fifth World Congress on Intelligent Control and Automation (WCICA 2004)*, 3, 2219 – 2221.

Xia, W., & Wu, Z. (2005). An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. *Computers & Industrial Engineering*, 48(2), 409-425.

Yin, P.Y., Yu, S.S., Wang, P.P., & Wang, Y.T. (2007a). Multi-objective task allocation in distributed computing systems by hybrid particle swarm optimization. Applied Mathematics and Computation, 184(2), 407-420.

Yin, P.Y., Yu, S.S., Wang, P.P, & Wang, Y.T. (2007b). Task allocation for maximizing reliability of a distributed system using hybrid particle swarm optimization. *The Journal of Systems & Software*, 80(5), 724-735.

Zhang, H., Li, X., Li, H., & Huang, F. (2005). Particle swarm optimization-based schemes for resource-constrained project scheduling. *Automation in Construction*, 14, 393-404.

Zhang, H., Li, H., & Tam, C.M. (2006). Particle swarm optimization for resource-constrained project scheduling. *International Journal of Project Management*, 24(1), 83-92.

Zhi, X.H., Xing, X.L., Wang, Q.X., Zhang, L.H., Yang, X.W., Zhou, C.G., & Liang, Y.C. (2004). A discrete PSO method for generalized TSP problem. *Proceedings* *of 2004 International Conference on Machine Learning and Cybernetics*, 4, 2378-2383.

作者簡介 (Biography)

姓名(Name)：徐誠佑 (Cheng-Yu Hsu)

學歷：

專士	明志工專	工業工程與管理科	(80.9~85.7)
學士	台灣科技大學	工業管理系	(88.9~90.7)
碩士	清華大學	工業工程與工程管理學系	(90.9~92.7)
博士	交通大學	工業工程與管理學系	(92.9~96.6)

著作：

一、 已接受之期刊論文

1. D.Y. Sha, C.-Y. Hsu, 2007, "A new particle swarm optimization for the open shop scheduling problem," *Computers and Operations Research* (Accepted) (SCI).

2. D.Y. Sha, C.-Y. Hsu, 2006, "A hybrid particle swarm optimization for job shop scheduling problem," *Computers & Industrial Engineering*, Vol. 51, No. 4, pp. 791-808 (SCI).

二、 審查中期刊論文

1. D.Y. Sha, C.-Y. Hsu, "A new discrete binary particle swarm optimization for the multidimensional 0-1 knapsack problem," (submitted to *European Journal of Operational Research*) (SCI).

三、 研討會論文

1. D.Y. Sha, C.-Y. Hsu, 2006, "A particle swarm optimization with parameterized active schedules for the open shop scheduling problem," *Proceedings of the 2006 CIIE Annual Conference*.
2. D.Y. Sha, C.-Y. Hsu, 2006, "A modified parameterized active schedule generation algorithm for the job shop scheduling problem," *Proceedings of the 36th International Conference on Computers and Industrial Engineering (ICCIE 2006)*, pp. 702-712, Taiwan, ROC.